Fusion for object detection

By

Pan Wei

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Electrical and Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

August 2018

Fusion for object detection

By

Pan Wei

Approved:

John E. Ball
(Major Professor)

Nicolas H. Younan
(Committee Member)

Derek T. Anderson
(Committee Member)

Christopher J. Archibald
(Committee Member)

James E. Fowler
(Graduate Coordinator)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Pan Wei

Date of Degree: August 10, 2018

Institution: Mississippi State University

Major Field: Electrical and Computer Engineering

Major Professor: Dr. John E. Ball

Title of Study: Fusion for object detection

Pages of Study: 154

Candidate for Degree of Doctor of Philosophy

In a three-dimensional world, for perception of the objects around us, we not only wish to classify them, but also know where these objects are. The task of object detection combines both classification and localization. In addition to predicting the object category, we also predict where the object is from sensor data. As it is not known ahead of time how many objects that we have interest in are in the sensor data and where are they, the output size of object detection may change, which makes the object detection problem difficult.

In this dissertation, I focus on the task of object detection, and use fusion to improve the detection accuracy and robustness. To be more specific, I propose a method to calculate measure of conflict. This method does not need external knowledge about the credibility of each source. Instead, it uses the information from the sources themselves to help assess the credibility of each source. I apply the proposed measure of conflict to fuse independent sources of tracking information from various stereo cameras. Besides, I propose a computational intelligence system for more accurate object detection in real–time. The proposed

system uses online image augmentation before the detection stage during testing and fuses the detection results after. The fusion method is computationally intelligent based on the dynamic analysis of agreement among inputs. Comparing with other fusion operations such as average, median and non-maxima suppression, the proposed methods produces more accurate results in real-time. I also propose a multi–sensor fusion system, which incorporates advantages and mitigate disadvantages of each type of sensor (LiDAR and camera). Generally, camera can provide more texture and color information, but it cannot work in low visibility. On the other hand, LiDAR can provide accurate point positions and work at night or in moderate fog or rain. The proposed system uses the advantages of both camera and LiDAR and mitigate their disadvantages. The results show that comparing with LiDAR or camera detection alone, the fused result can extend the detection range up to 40 meters with increased detection accuracy and robustness.

Key words: Information fusion, object detection, computational intelligence, deep learning

DEDICATION

To my family.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Object detection in computer vision is the process of identifying and localizing certain categories of objects we are interested in. Object detection is widely used in autonomous driving, Advanced Driver Assistance Systems (ADAS), manufacturing industry, security, and so on. For example, ADAS assist drivers by providing advice and warnings when necessary. ADAS includes a subsystem of object detection that helps detecting other cars, pedestrians, traffic signs like speed limit signs, and so on. Another interesting application is Amazon Go store. In this store, there are no checkout lanes. They use sensors to detect what items customers take off the shelves and put into their cart. After customers walk out of the door, the system automatically charges their Amazon account.

For object detection, we need not only classify the object but also localize the object. There are several tasks for image object detection. First, we need to detect as many objects we have interest in as possible. Second, we need to give accurate labels to each detected object. Third, we need to point out the accurate position of each object in the image. In other words, we need to draw the bounding boxes accurately including exactly the whole part of the objects.

To achieve these goals, multiple methods [85, 86, 87, 97, 13, 61], especially ones based on deep learning have been utilized. Each of these methods has its advantages and disadvantages. For example, YOLO [85, 86], which is short for You Only Look Once, is known for its fast processing speed, but is unable to detect small object in some scenarios. On the other hand, Faster R-CNN [87] may lack the speed for real-time applications but performs better in accuracy in some scenarios like smaller objects. Also, a detector with one set of parameters is usually not enough. For example, in order to detect tiny faces, some researchers trained another set of parameters separately [44]. Therefore, one natural idea that comes to mind for improving detection accuracy and robustness is to combine/fuse detection results using different methods or parameters.

For object detection, besides cameras, we can also use other sensors, such as a LiDAR, a Radar, an ultrasonic sensor, and so on. Each type of sensor provides different type of data. Similarly, each type of sensor has its own advantages and disadvantages. For example, LiDAR can accurately obtain the position information of the object detected, but it does not have much information about the type of object it detects, and it cannot "see" in color. The only other information besides position about each point detected by LiDAR is the reflecting intensity. One the other hand, a single camera or multiple cameras cannot accurately tell the position of the object compared to a LiDAR, but by using the texture and color information of the detected object, it can help classifying the detected object better than LiDAR. As a result, another idea for improving detection is by fusing detection results from multiple types of sensors.

In order to achieve more robust and accurate object detection results, the two approaches I developed on are fusion of multiple inputs via various methods or sensors. Then, the question remains to be solved is how to fuse multiple inputs. The fusion methods I am applying are either computationally intelligence based or learning based. First, to fuse inputs together in a computational intelligent way, my idea is to utilize fuzzy-integral-based methods. For these type of methods, the inputs are fused together dynamically, not statically, which means that it utilizes certain measures that could exact certain properties from inputs themselves. Another fusion idea is by learning. This means that with plenty of training samples, by using machine learning methods, we can learn certain features to fuse inputs.

In summary, I develop on the fusion of multiple inputs for better object detection results. In this process, I utilize various detection methods and sensors for diverse inputs, and fuse these inputs using computational-intelligence based or learning-based methods.

1.2 Contributions

The aim for my research is to give "better" detection results than results before fusion. Herein, better refers to improvement on accuracy of localization or classification for detection.

Specifically, the contributions of my research are:

- Developed a measure to represent the conflicts among different sources.
- Utilized the proposed measure for interval-valued information fusion.

- Proposed a real-time detection fusion system, which utilizes image augmentation before the detection stage and fusion after to increase detection accuracy and robustness.
- Proposed a real-time detection fusion system using multiple sensors incorporating advantages and mitigate disadvantages of each type of sensor. The sensor types include LiDAR (Light Detection and Ranging) and camera.

1.3 List of Publications

The research in this dissertation is related to the following publications:

- Measuring Conflict in a Multi-Source Environment as a Normal Measure:

  [104] **P. Wei**, J. E. Ball, D. T. Anderson, A. Harsh, C. Archibald, "Measuring Conflict in a Multi-Source Environment as a Normal Measure," *IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, December, 2015.

- Multi-Sensor Conflict Measurement and Information Fusion:

  [102] **P. Wei**, J. E. Ball, D. T. Anderson, "Multi-Sensor Conflict Measurement and Information Fusion," *SPIE Defense, Security, and Sensing*, April, 2016.

- Fusion of an ensemble of augmented image detectors for robust object detection

  [103] **P. Wei**, J. E. Ball, D. T. Anderson, "Fusion of an ensemble of augmented image detectors for robust object detection," *Sensors*, 18(3), 894, March, 2018.

- Detection fusion in an industrial multi-sensor collision avoidance system

  P. Wei, L. Cagle, T. Reza, J. E. Ball, J. Gafford, "Detection fusion in an industrial multi-sensor collision avoidance system," submitted, 2018.

1.4 Outline

Chapter 1 gives an introduction to the problem of using fusion for object detection. Moreover it introduces my contribution to this field and publications related to this dissertation.

In Chapter 2, I present basics for previous approaches to fusion using computational intelligence and to detection using deep learning.

In Chapter 3, I propose an algorithm to measure conflict in interval-valued settings using a normal measure. In the synthetic and experimental examples, the results show that the output is similar to our "common sense", and can help in identifying conflicting sources.

In Chapter 4, I utilize the proposed Conflict Measure (CM) in Chapter 3 to fuse 3D tracking information. The results show that conflict measure can be used as a way to calculate source credibility, and fuse different sources based on this credibility.

In Chapter 5, I propose an object detection system, which uses online image augmentation before the detection stage and fusion after to increase detection accuracy and robustness. The proposed fusion method is a computationally intelligent approach, based on dynamic analysis of agreement among inputs for each case (fuzzy measure of agreement).

In Chapter 6, I propose a multi–sensor detection system that fuses both the camera and LiDAR detection to obtain a more accurate and robust beacon detection. The proposed fusion system is utilized in an industrial collision avoidance system, and tested under various circumstances.

Finally, Chapter 7 gives conclusion of this dissertation with ideas for future work.

CHAPTER 2

LITERATURE REVIEW

In this chapter, basic concepts related to computationally intelligent fusion including fuzzy measure, Choquet integral (ChI), interval-valued fuzzy integral and fuzzy measure of agreement are listed. Besides that, we can also use neural networks (deep learning) related methods to do detection. Therefore, this chapter briefly summarizes the history and development of deep learning, two types of neural networks (multilayer perceptron and convolutional neural networks). I also present a short introduction of object detection using deep learning.

2.1 Basics for computational intelligence approach to fusion

The following includes some basic concepts for computational intelligence fusion approaches.

2.1.1 Fuzzy Measure

The ChI is defined with respect to the Fuzzy Measure (FM). The FM is the modeling of the worth of the individual inputs and their various interactions. Ultimately, once we select an FM, the ChI becomes a specific operator.

Let $X = \{x_1, ..., x_n\}$ be a set of n information inputs (e.g., experts, sensors, algorithms, etc.), where x_i represents the i-th input. For example, in object detection in images, x_i is the i-th Axis-Aligned Bounding Box (AABB). The definition of FM is as follows,

Definition 2.1.1 (Fuzzy measure) *For a finite set X, a measure g: $2^X \rightarrow$ [0,1] is an FM if it has the following properties:*

1. *(Normality) $g(\emptyset) = 0$,*
2. *(Monotonicity) If $A, B \in 2^X$ and $A \subseteq B \subseteq X$, then $g(A) \leq g(B) \leq 1$.*

Note, often, $g(X) = 1$ for problems like confidence/decision fusion. Property 2 of the FM states that g is monotone. For example, if there are three inputs $\{x_1, x_2, x_3\}$, then the FM $g(\{x_1, x_2, x_3\}) \geq max\left(g(\{x_1, x_2\}), g(\{x_1, x_3\}), g(\{x_2, x_3\})\right)$ and $g(\{x_1, x_2\}) \geq max(g(\{x_1\}), g(\{x_2\}))$. What this means is adding inputs never decreases the measure.

The FM (g) lattice (a graph that shows the variables and the minimal set of required monotonicity constraints as edges) for three inputs ($N = 3$) is shown in Figure 2.1. In the first layer, the variables are only for a single input. In the second layer, variables are for two inputs, and so forth.

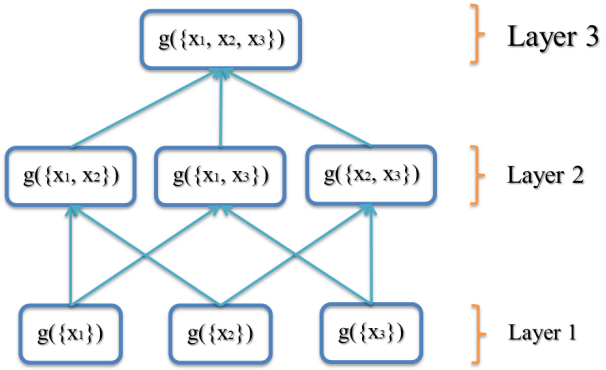


Figure 2.1

Fuzzy measure shown for three information inputs.

2.1.2 Fuzzy Measure of Agreement

The FM can be (i) specified by an expert, (ii) learned from the training data, (iii) input from densities or (iv) extracted from the current observation (this paper). One example to extract the FM from the current observation is the FM of agreement proposed in [100, 37, 36].

The motivation for the proposal of the FM of agreement is that sometimes, experts' estimation or the ground-truth can be challenging to obtain, and all we have is the data themselves. This means that we have the inputs, but we do not know their "worth". How the FM of agreement decides the "worth" of each input is by calculating the amount of agreement between each input and other inputs. Herein, agreement means the amount of overlap among different inputs. In the extreme case, if one input has no overlap with any other inputs, the FM of agreement for two-tuples is zero.

The following demonstrates the way to calculate the FM of agreement when the available evidence $\bar{h} = \{\bar{h}_1, ..., \bar{h}_n\}$ is interval-valued.

Let $\bar{A}_i = \{\bar{h}_{\pi(1)}, ..., \bar{h}_{\pi(i)}\}$ be the permuted set of intervals and z_i be the weight of each term, where $z_2 \leq z_3 \leq ... \leq z_n$. In [100], $z_i = i/n$ and:

$$\tilde{g}^{AG}(\bar{A}_0) = \tilde{g}^{AG}(\bar{A}_1) = 0, \tag{2.1a}$$

$$\begin{aligned}\tilde{g}^{AG}(\bar{A}_i) = &\left| \bigcup_{k_1=1}^{i-1} \bigcup_{k_2=k_1+1}^{i} \bar{h}_{\pi(k_1)} \cap \bar{h}_{\pi(k_2)} \right| z_2 \\ &+ \left| \bigcup_{k_1=1}^{i-2} \bigcup_{k_2=k_1+1}^{i-1} \bigcup_{k_3=k_2+1}^{i} \bar{h}_{\pi(k_1)} \cap \bar{h}_{\pi(k_2)} \cap \bar{h}_{\pi(k_3)} \right| z_3 \\ &+ ... + |\bar{h}_{\pi(1)} \cap \bar{h}_{\pi(2)} \cap ... \cap \bar{h}_{\pi(i)}| z_i,\end{aligned} \tag{2.1b}$$

$$g^{AG}(\bar{A}_i) = \frac{\tilde{g}^{AG}(\bar{A}_i)}{\tilde{g}^{AG}(\bar{h})}, \quad i = [2:n]. \tag{2.1c}$$

Equation (2.1a) means that there is no "worth" for the empty and one-element set, because we need at least two elements to compare.

2.1.3 Choquet Integral

Many variants of the fuzzy integral have been put forth to date [35], e.g., the Sugeno FI [95], the ChI [76], etc. Herein, we use the ChI, as it produces results anywhere between the minimum and maximum inputs. The Sugeno FI only yields one of the N inputs or 2^N FM variable values due to its min/max formulation.

Definition 2.1.2 ***(Choquet integral)*** *Let π be a permutation of X, such that $h(x_{\pi(1)}) \geq h(x_{\pi(2)}), ..., \geq h(x_{\pi(n)})$, $A_i = \{x_{\pi(1)}, ..., x_{\pi(i)}\}$ and $g(A_0) = 0$. Let $h : X \rightarrow \Re$ be a real-valued function that represents inputs. The ChI is*

$$\int_c h \circ g = C_g(h) = \sum_{i=1}^{n} h(x_{\pi(i)})[g(A_i) - g(A_{i-1})]. \tag{2.2}$$

2.1.4 Interval-valued fuzzy integral

In some cases, evidence h is not represented as a single value, but as an interval. Intervals capture uncertainty at a level of granularity that is acceptable for many applications.

Definition 2.1.3 ***(Interval-valued fuzzy integral)*** *Let $\bar{h}_i \subseteq I$ be the continuous interval-valued evidence from input x_i and let $[\bar{h}_i]^- \leq [\bar{h}_i]^+$ be the left and right endpoints of interval $\bar{h}_i$ respectively. The interval-valued FI is [35]*

$$\int \bar{h} \circ g = [\int [\bar{h}]^- \circ g, \int [\bar{h}]^+ \circ g]. \tag{2.3}$$

This equation means that interval-valued input breaks down into the extreme (interval boundaries) case. This is the same for set-valued input as well. In [3], there is a review of different extensions of the FI for uncertain h and/or g.

2.2 Basics for deep learning approach to detection

For both detection and fusion, deep learning methods are used in my research. The following introduces history, development, and some basic concepts of deep learning.

2.2.1 History and Development

Everything has a start. For Neural Networks (NN) (later named as Deep Learning), it starts in 1943, when the first mathematical model of neural network was proposed in reference [72] by McCulloch and Pitts.

One of the basic NN architectures is the perceptron. In 1957, Frank Rosenblatt proposed the perceptron algorithm [88]. The Mark I Perceptron was the first implementation of this algorithm. In 1969, Marvin Minsky and Seymour Papert published the book "Perceptrons: An Introduction to Computational Geometry" [75]. In this book, they pointed out two problems of perceptrons: One is that single-layer neural network cannot solve the linearly unsolvable problems, like XOR; The other is the huge amount of calculation which cannot be performed by computers at that time. This book is claimed to be responsible for the AI winter until the 1980s.

In 1986, David Rumelhart, Hinton Geoffrey, and Williams Ronald published the paper [90]. This paper systematically explained the use of back-propagation on NN. Using back-propagation with one hidden layer, the XOR problem is solved, and its calculation is proportional to the number of neurons, which is implementable using computers at that time. This is the beginning of the NN revival. From then on NNs have been used in various areas. In 1989, Yann LeCun published a paper [52], which applied back-propagation to recognize handwritten zip codes. The error rate is said to be 5%. In the 1990s, LeCun developed a convoluted neural network technique [51], which is applied to recognize hand-written digits on checks.

In the meantime, NN has faced many problems. One major problem is the long training time for a simple NN. The problems are largely alleviated with the appearance of general purpose graphics processing unit (GPU). Unlike the single instruction multiple data structure of central processing unit (CPU), GPU can do parallel and batch processing, which is much faster than the traditional CPU.

What really escalated the renaissance of DL is AlexNet's [1] winning of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [17] in 2012. The team used two Nvidia GTX 580 GPUs and trained using 1.2 million images in six days. This is the first time in twenty years that NN largely won over other techniques. From then on, DL based networks, such as ZF Net [107], VGG Net [94], GoogleLeNet [96], and ResNet [39], won ILSVRC every year.

In recent years, DL has grown into full blossom. Five major top technology companies - Facebook, Amazon, DeepMind/Google, IBM and Microsoft come together and create partnership on AI. DL based Go-playing AI AlphaGo [93] has beaten the best Go-player in the world. The latest version of Go-playing AI AlphaGo Zero [92] is significantly better than the previous version, and more importantly it is entirely self-taught.

2.2.2 Neural Networks

For neural networks, it starts with the concept of neuron, and has been developed into different architecture branches. In the following, the basic neuron model is introduced, and two commonly used neural network structure: multilayer perceptron, and convolutional neural networks.

2.2.2.1 Neuron model

NN is inspired from the biological neural network in brain. In a simple model of the brain, it consists of neurons connected by synapses. Each neuron receives information from its dendrites and sends out signal along axons. Each axon is connected with other neuron's dendrites via its terminals [55]. In this process, signals are received by the neuron. When the received signal passes a certain voltage threshold, it is "activated" and the signal is passed on to the next neuron. But the real brain functioning is extremely complicated than this simple model, and people has proposed more complicated models describing neuron's dynamics and computation [6].

Similarly, the basic computation unit in NN is also called neuron. In [72], the above description of neuron is summarized as a model shown in Figure 2.2. In this model, the neuron receives n signals from other neurons via weighted connections. The sum of the weighted inputs signal plus a constant goes through activation function f and produces output y.

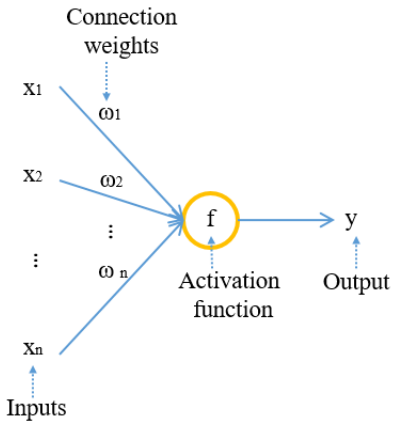


Figure 2.2

Neuron Model.

For activation functions, there are various forms. In history, one of the most commonly used one is called the Sigmoid function. Its mathematical form is $f(x) = \frac{1}{1+e^{-x}}$. This function "squashes" the output value to a range between 0 and 1. Nowadays, in deep neural networks, The Sigmoid function has been replace by other functions like the Rectified Linear Unit (ReLU) [77], with the mathematical form as $f(x) = max(0, x)$, because they are so efficient to calculate and are still effective. There are other variations of activation functions, such as the Leaky ReLu [64], the exponential linear unit (ELU) [12], the scaled exponential linear units (SELUs) [49], and so on.

2.2.2.2 Multilayer perceptron

One of the simplest NN structures is the multilayer perceptron (MLP). The MLP is a feedforward NN with at least three layers of neurons, which are input layer, hidden layer and output layer as shown in Figure 2.3.

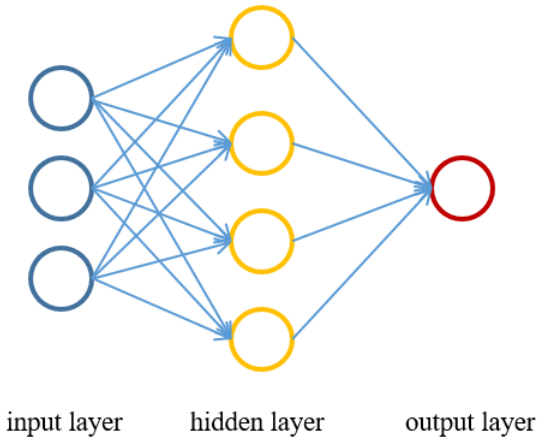


Figure 2.3

One simple example of multilayer perceptron (MLP).

The input layer receives the inputs from outside sources, and these inputs are processed via hidden layers. The final results are given out through output layer. The input layer only receives not processing, while the hidden layer and output layer function more like neuron doing signal processing. So, Figure 2.3 is always called a two-layer neural network. Here, the data in the input layer does not undergo convolution, and this is a good example of a "shallow" NN.

During the training of a MLP, what we need to decide first is the network structure, initial weights, learning rate, and loss function. Then by using certain learning algorithm, such as Back Propagation (BP) [90], the NN can adapt and learn the weights. During testing, what we expect is that by using the weights obtained from training, a new input goes through the network and produces an output that is as close as possible to the ground-truth. For example, let us assume that we are facing a classification problem to find a cat. We first use multiple cat and non-cat images to train a MLP. When given a new image which contains a cat, a well-trained MLP would output a high classification score on the category cat.

As BP is one of the most successful learning algorithm, herein I introduce how BP works through a simple example.

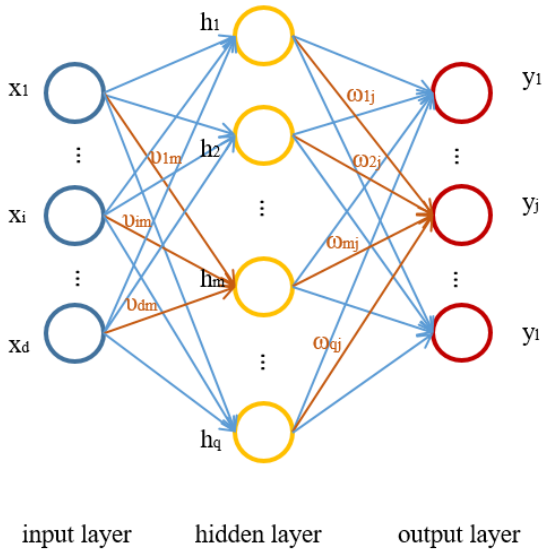


Figure 2.4

One example showing how back propagation(BP) works.

Given a training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ with m instances, where $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^l$. The network structure we design is shown in Figure 2.4. This network has d inputs neurons, q hidden layer neurons and l output layer neurons. The weight between the i^{th} input neuron and the m^{th} hidden layer is υ_{im}. The weight between the m^{th} hidden neuron and the j^{th} output neuron is ω_{mj}. The bias for the m^{th} hidden neuron is γ_m. The bias for the j^{th} output neuron is θ_j. Let the activation function be a sigmoid. For a training sample pair $(\mathbf{x}_k, \mathbf{y}_k)$, let the output from the NN be $\hat{\mathbf{y}}_k = (\hat{y}_1^k, \hat{y}_2^k, ..., \hat{y}_l^k)$. Also, let $\beta_j = \sum_{m=1}^{q} \omega_{mj} h_m$.

$$\hat{y}_j^k = Sigmoid(\beta_j + \theta_j). \tag{2.4}$$

To calculate the distance between the prediction using this MLP and ground-truth, we use standard deviation as follows:

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2. \tag{2.5}$$

To adjust weight ω, we use gradient descent with η as the learning rate, which is:

$$\Delta\omega_{mj} = -\eta\frac{\partial E_k}{\partial \omega_{mj}} \tag{2.6}$$

From the characters of Sigmoid function and partial derivation, we can have the update equation for ω_{mj} as:

$$\Delta\omega_{mj} = \eta\hat{y}_j^k(1-\hat{y}_j^k)(y_j^k - \hat{y}_j^k)h_m. \tag{2.7}$$

In the similar way, we can obtain the update function for other weights and bias.

In [40], authors prove that multilayer feedforward networks are universal approximators, which means that MLP can approximate any continuous function to any accuracy desired. But how to set the hidden layer is still an open question. What is usually used in practice is typically empirically determined (e.g. by trial-and-error). The learning rate can be adjusted and decays over time. We can also try different initialization, regularization and architecture.

2.2.2.3 Convolutional Neural Networks

Besides the MLP, there is another widely used neural network is called convolutional neural networks (CNN). CNNs have mainly been applied to imagery. An early form of CNN is called LeNet-5 [53] used for handwritten digits recognition. CNN has been used

in various applications, such as classification [50, 96, 39], detection [85, 86, 87], instance segmentation [38], and so on.

A typical CNN architecture usually has five basic building blocks, which are input layer, convolutional layer, ReLu layer, pooling layer and fully-connected layer.

The usually input layer is 3-dimensional, with each dimension representing red, green or blue channel of a RGB image. The input layer represents the input image. For example, for a 32×32 input image, the input layer is shown in Figure 2.5.

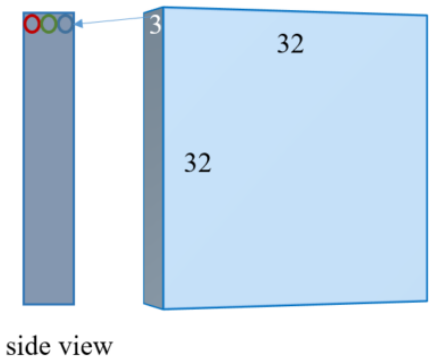


Figure 2.5

One example of input layer for CNN.

The core building block for CNN is a convolutional layer, which consists of several dot product filters. For example, suppose the input size before the convolutional layer is 32×32×5. Assume there are 12 filters, and that each filter size is 3×3×5. Stride by which the filter shifts is 3, without any zero padding, which pads the input with zeros around the

border. Based on the following equation, where W stands for the input height/width, F stands for the filter height/width, P is the zero padding amount, S is the stride amount.

$$\frac{W - F + 2P}{S} + 1 \tag{2.8}$$

The output size after convolution is 32×32×12 as shown in Figure 2.6.

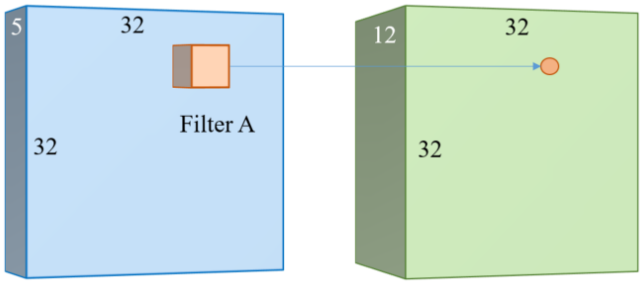


Figure 2.6

One example of how convolutional layer works in CNN.

The ReLu layer uses function $max(0, x)$ for each neuron. This layer will not change the size but will make all values equal or larger than 0. The pooling layer is doing down-sampling with operation max or $average$. Pooling will change the size (width and height). Fully-connected layer is connected to all neurons in the previous layer, which is the same as in regular neural networks. One simple example is shown in Figure 2.7.

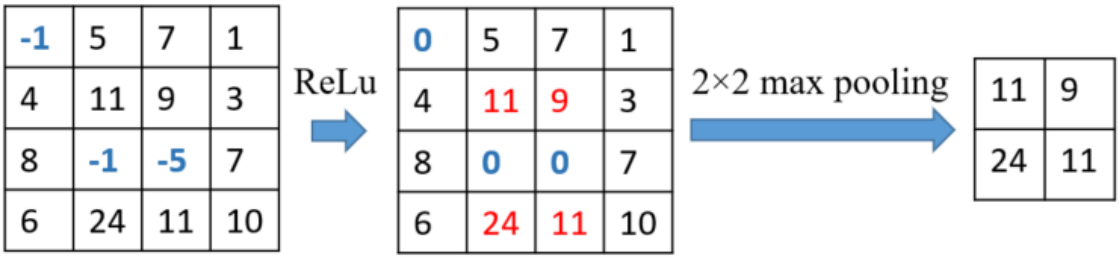


Figure 2.7

One example of how ReLu layer and pooling layer work in CNN.

2.2.3 Object Detection

Due to the development of deep learning, the accuracy for object detection in images has largely been improved. Examples of these object detectors includes: YOLO [85, 86], Faster R-CNN [87], Multibox [97], Region-based Fully Convolutional Networks (R-FCN) [13] and Single Shot MultiBox Detector (SSD) [61]. These detectors surpass traditional methods such as histogram of oriented gradients [14], deformable parts model (DPM) [25] in both accuracy and speed.

There are basically two types of detection methods using deep learning. One type is based on region proposals, and one example is Faster R-CNN [87]. For this type of method, the network is divided into two parts, and the first part is mainly used to propose possible regions for detection. The other type of detection method detects without proposing possible regions. Instead, it localizes and classifies objects using one giant convolutional network. One typical example for these detectors is called YOLO [85] [86]. YOLO uses a single neural network for simultaneously predicting the location of the object, confidence score for the location and classification scores.

CHAPTER 3

MEASURING CONFLICT IN A MULTI-SOURCE ENVIRONMENT AS A NORMAL MEASURE

3.1 Introduction

Living in an imperfect world, we can never fully trust a source due to various reasons such as noise, faulty sensors, deception, etc. In some scenarios we do not have external knowledge about the credibility of each source (in other words, how much we can trust each source), the only information we have is the information from the sources. Herein, we focus on measuring conflict from source information to help assess credibility. In many applications, supposing each source is independent, we can use such a measure to help identify conflict within our sources. In order to facilitate better decisions, we can put more trust on those sources who have less conflict and diminish the influence of the sources who have conflict above some amount. However, we do note that in some cases conflicting cases can be the most interesting and deserve further analysis.

Herein, we focus on a new simple method to calculate measure of conflict. Some theories exist to measure conflict: Shannon entropy [4], fuzzy measure [29] and belief theory [68]. On the other hand, a number of fuzzy measures of agreement have been put forth to date [34, 108, 100, 37, 36]. While related, conflict and agreement can be quite

different and difficult to describe in terms of one another. Conflict is a challenge to capture in terms of an algorithm that is "in line", with what a human expects.

3.2 Background

To better understand the proposed measure of conflict, we first need to review the following definitions of a normal measure (NM). Let $X = \{x_1, ..., x_n\}$ be a set of n information sources (e.g., experts, sensors, algorithms, etc.).

Definition 3.2.1 ***(Normal measure)*** *[27] Let (X, Ω) be a measurable space, where X is a set and Ω is a σ-algebra of X. A measure g: $\Omega \rightarrow$ [0, 1] is a NM if there exists a minimal set A_0 (e.g., $\emptyset$) and a maximal set A_m (e.g., X) in Ω such that:*

1. $g(A_0) = 0$,
2. $g(A_m) = 1$.

The measures calculated using the algorithm proposed in this chapter is a NM. The reason for using a NM is that if we used a fuzzy measure then adding a conflicting source could lessen our measure value; however, it cannot due to FM monotonicity.

3.3 Measure of Conflict

In order to measure and identify conflict, we propose an algorithm called the measure of conflict (CF). The CF is defined here as the sum of sub-interval conflicts over the all sources' measurement range. Each sub-interval contributes to the conflict if some of the sources' measurement range does not cover such sub-interval. The conflict metric is the ratio between the sub-interval length and the global measurement range length multiplied

by the percentage of sources not covering that sub-interval. The nomenclature and mathematical description of the algorithm is as follows:

Let $X = \{x_1, ..., x_n\}$ be a set of sources for which we do not know the credibility of each source. Each source x_i provides interval-valued evidence ($\overline{h}_i$). For example, if sensor x_1 has output voltages v, where $2.0 \leq v \leq 3.5$, then the interval is $\overline{h}_1 = [2.0, 3.5]$, and this is called interval-valued evidence. A_i is the set that contains all i-tuple combinations of the sources, where i is the number of sources in the tuple. For example, with sources x_1, x_2 and x_3, $A_1 = \{\{x_1\}, \{x_2\}, \{x_3\}\}$, $A_2 = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}\}$, and $A_3 = \{x_1, x_2, x_3\} = X$. g is the numeric NM.

Suppose there are n interval-valued sources. Let $E = \{E_1, ..., E_{2n}\}$ be the ordered set ($E_i \leq E_{i+1}$) of all interval endpoints from our evidences. An example is shown in Figure 3.1. Let $\overline{P} = \{\overline{P}_1, ..., \overline{P}_{2n-1}\}$ be the set of intervals induced by E. For example, $\overline{P}_1 = [E_1, E_2]$. Let $O(\overline{P}_k)$ be the number of sources (which are in $A' \in A_i$) in interval $\overline{P}_k$. Let g^{CF} be defined as:

$$g^{CF}(A' \in A_1) = 0, \forall A' \in A_1 \tag{3.1}$$

$$\tilde{g}^{CF}(A' \in A_i) = \sum_{k=1}^{2n-1} \Phi(\overline{P}_k, A') |\overline{P}_k| \left(\frac{i - O(\overline{P}_k)}{i} \right),$$

$$\Phi(\overline{P}_k, A') = \begin{cases} 1 & if\ \overline{P}_k \subseteq [\min_{x_j \in A'} [\overline{h}_j]^-,\ \max_{x_j \in A'} [\overline{h}_j]^+], \\ 0 & else, \end{cases}$$

$$i = [2 : n], \tag{3.2}$$

$$g^{CF}(A_i) = \tilde{g}^{CF}(A_i)/(\max_{x_j \in A'}[\overline{h}_j]^+ - \min_{x_j \in A'}[\overline{h}_j]^-). \tag{3.3}$$

In CF, every interval that is bounded within the maximum right and minimum left endpoints have been weighted based on the number of overlap times. If an interval has no sources in it then it has the highest conflict weight of one. On the other hand, if all sources overlap in an interval then there is no conflict and a weight of zero is assigned. The CF not only considers intervals that have overlapping sources, but also considers all intervals without any overlapping sources (which is the section marked as 0 in Figure 3.1). Using CF, the differences between case 1 and case 2 in Figure 3.2(a) and Figure 3.2(b) can be clearly shown. In Figure 3.2(a), comparing the two cases, there is more conflict among the three sources of case 1 than that of case 2 since the interval from source 1 is more similar to source 2 and therefore there is a wider overlapping region among the three sources. In Figure 3.2 (b), case 2 has more conflict between the two sources since there is a wider no overlapping region versus case 1. These two examples show that CF works "in line" with what a human would expect.

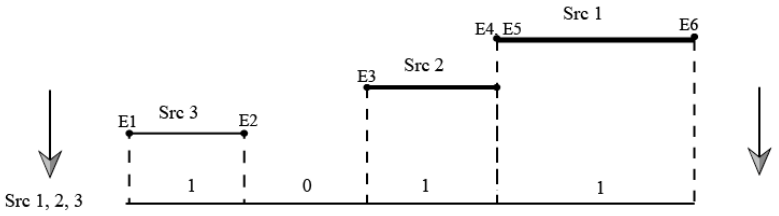
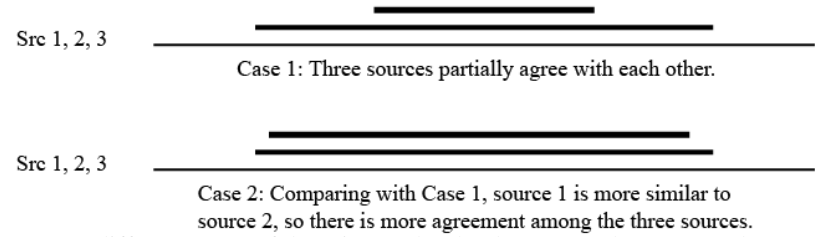
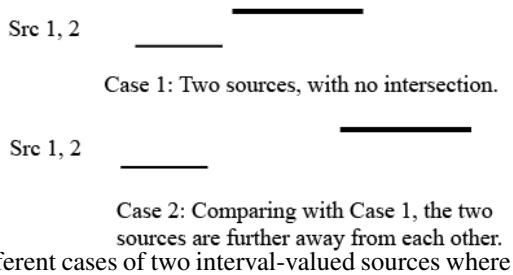


Figure 3.1

Counting the number of sources with one region that has no sources and is marked as 0.



(a) Two different cases of three interval-valued sources where Case 1 has more conflict than Case 2.



(b) Two different cases of two interval-valued sources where Case 2 has more conflict than Case 1.

Figure 3.2

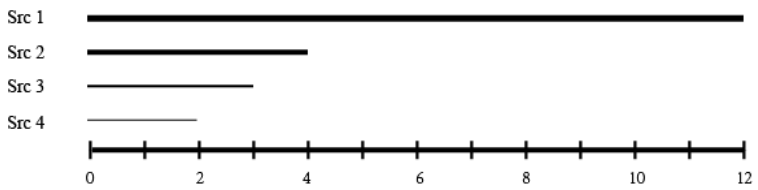
Cases with different conflict.

3.4 Numeric Examples

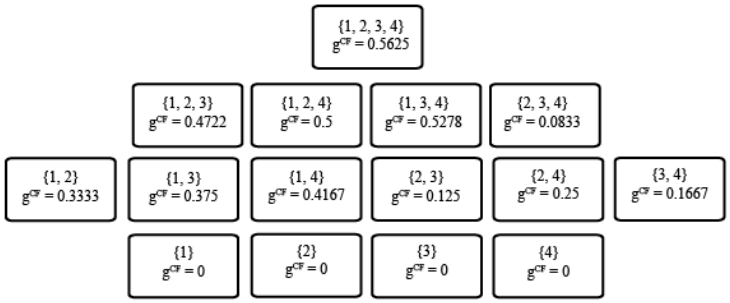
3.4.1 Synthetic Examples

In this section, three synthetic examples are provided to demonstrate the calculation and meaning of CF for increasing degrees of conflict.

Example 1. In the first example, there are four interval-valued evidences, $\overline{h}_1 = [0, 12]$, $\overline{h}_2 = [0, 4]$, $\overline{h}_3 = [0, 3]$, and $\overline{h}_4 = [0, 2]$. The four evidences are shown in Figure 3.3(a), while the lattice of CF conflict measures is shown in Figure 3.3(b). The following is a manual calculation example using CF.



(a) Four interval-valued sources.



(b) Lattice of CF values.

Figure 3.3

Example 1: Small conflict.

For the calculation of $g^{CF}(\{x_1, x_2, x_3\})$, as shown in Figure 3.3, interval [4, 12] has one source, which is source 1. Interval [3, 4] has two overlapping sources which are from source 2 and 3. Interval [0, 3] has all three overlapping sources. We can compute CF using Equation 3.2 as follows:

$$\tilde{g}^{CF}(\{x_1, x_2, x_3\}) = |12 - 4| \times \tfrac{3-1}{3} + |4 - 3| \times \tfrac{3-2}{3} = \tfrac{17}{3},$$

$$g^{CF}(\{x_1, x_2, x_3\}) = \tfrac{17}{3} / |12 - 0| \approx 0.4722.$$

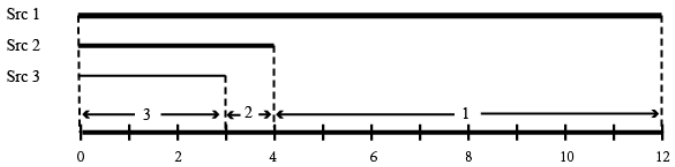


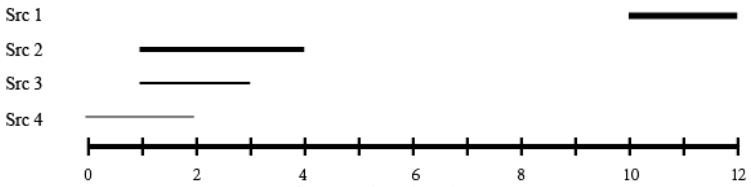
Figure 3.4

An example showing how to calculate $g^{CF}(\{x_1, x_2, x_3\})$ of three sources.

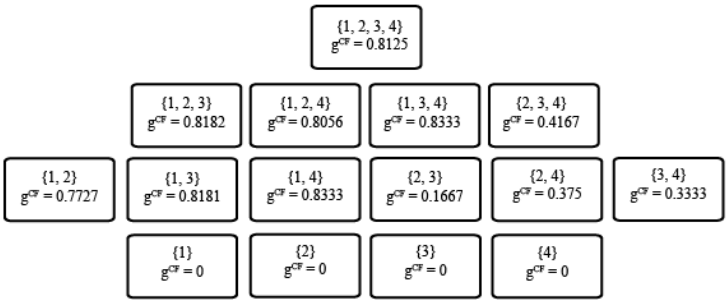
The results in Figure 3.3(b) show that CF is similar to what many of us would expect in terms of conflict. For example, the algorithm gives higher conflict measure to $\{x_1, x_2, x_4\}$ than $\{x_1, x_2, x_3\}$. This is because source 3 has more overlap than source 4, to sources 1 and 2.

Looking through the CF measures, we can find that the highest difference happens between $g^{CF}(\{x_1, x_2, x_3, x_4\})$ and $g^{CF}(\{x_2, x_3, x_4\})$ without considering the lowest layer. This shows that by adding source 1, the conflict measure rises the most, which means that source 1 can be identified as the most conflicting source.

Example 2. In the second example, the four interval-valued evidences are $\overline{h}_1 = [10, 12]$, $\overline{h}_2 = [1, 4]$, $\overline{h}_3 = [1, 3]$, and $\overline{h}_4 = [0, 2]$. The four evidences are shown in Figure 3.5(a), while the lattice of CF conflict measures is shown in Figure 3.5(b).



(a) Four interval-valued sources.



(b) Lattice of CF values.

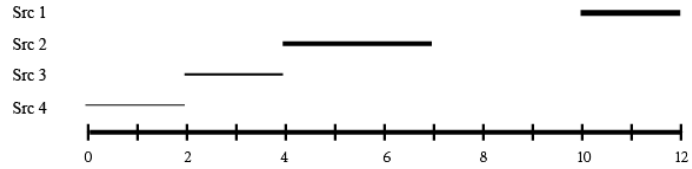
Figure 3.5

Example 2: Moderate conflict.

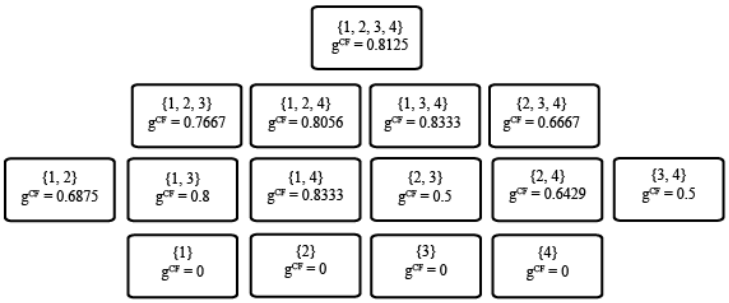
This example includes a unique source, i.e. source 1, which has no overlap with the other sources. Source 3 and 4 has the same length, and source 4 is further away from source 1 than source 3. Using CF, the measure value for $\{x_1, x_4\}$ is higher than the value for $\{x_1, x_3\}$, which is in accord with our expectation. The results in Figure 3.5(b) show that CF can well present the conflict among sources when there is unique source presented.

Similar to Example 1, the highest measure value difference also happens between $g^{CF}(\{x_1, x_2, x_3, x_4\})$ and $g^{CF}(\{x_2, x_3, x_4\})$ without considering the lowest layer. This difference helps us to identify source 1 as the unique (i.e. conflicting) evidence source.

Example 3. In the third example, there are four interval-valued evidences, $\overline{h}_1 = [10, 12]$, $\overline{h}_2 = [4, 7]$, $\overline{h}_3 = [2, 4]$, and $\overline{h}_4 = [0, 2]$. The four evidences are shown in Figure 3.6(a), while the lattice of CF conflict measures is shown in Figure 3.6(b).



(a) Four interval-valued sources.



(b) Lattice of CF values.

Figure 3.6

Example 3: Extreme conflict.

In this example, all four interval-valued evidences have no overlap with each other, which means this scenario is highly conflicted. It can be seen that all CF values in the upper three layers are higher than 0.5 and reach up to 0.8333. Comparing the three examples, every corresponding measure value in Example 1 and Example 2 is lower than or equal to the value in Example 3.

3.4.2 Experimental Example

Example 4. In this section, we use data collected from four temperature sensors as our sources. As shown in Figure 3.7, the sensors collect data once per second for 90 seconds. During this period, the temperature detected by sensor 1 rises due to an external influence. We select five seconds for our time interval as a sliding window and choose the maximum and minimum temperatures during the five-second period as the upper and lower endpoints for the intervals for each sensor. The five-second interval is chosen for convenience, and other choices would work as well. In Figure 3.8, the beginning of x axis is 5, which means that the related conflict measure is calculated using the period from 0 to 5 seconds. The measure used in Figure 3.8 is $g^{CF}(\{x_1, x_2, x_3, x_4\})$.

It can be seen that as the first temperature sensor output rises, the CF value also rises. It shows that values calculated using CF can be used as an indication of changes among sources.

In Figure 3.9, the measure used is $g^{CF}(\{x_2, x_3, x_4\})$, which is more stable than the values shown in Figure 3.8. This shows that sensor 1 is the most conflicting source among the four sensors.

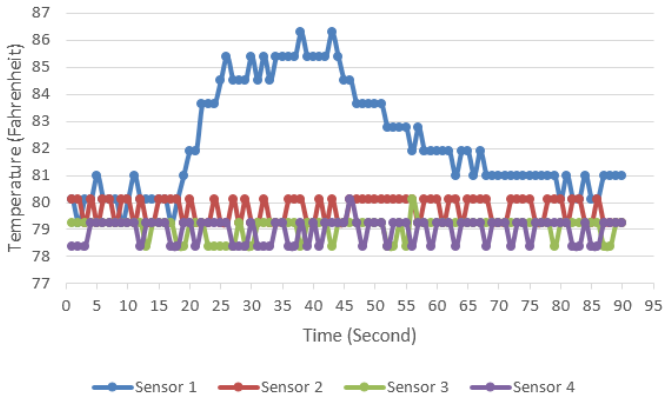


Figure 3.7

Experimental example: Measurement from four temperature sensors.

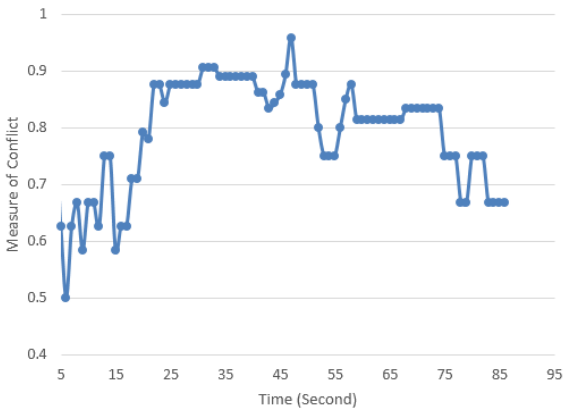


Figure 3.8

Experimental example: Measure values of $g^{CF}(\{x_1, x_2, x_3, x_4\})$.

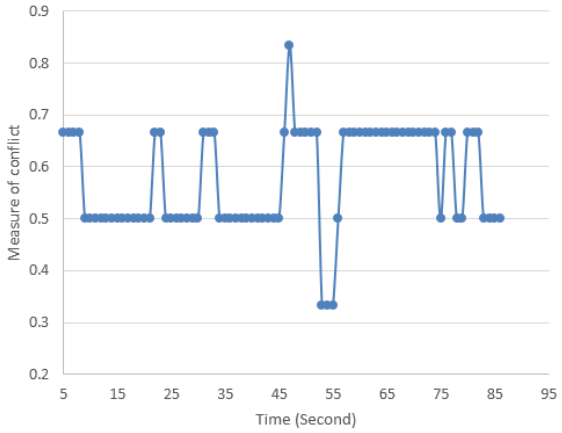


Figure 3.9

Experimental example: Measure values of $g^{CF}(\{x_2, x_3, x_4\})$.

3.5 Conclusions

In this chapter, I developed a measure of conflict in interval-valued settings as a normal measure. The results show that the output from CF has similar meaning to our "common sense", and it can help to identify conflicting sources.

CHAPTER 4

MULTI-SENSOR CONFLICT MEASUREMENT AND INFORMATION FUSION

4.1 Introduction

In classification or clustering, one is predicting or learning "structure" (patterns) in data or identifying anomalies, which are examples that do not fit the "norm". In this chapter, we discuss an algorithm that measures the regions of overlap, that is, regions where the multiple sensors viewing the same scene have outputs that agree. We call the output of this algorithm the conflict measure (CM). These regions are based solely on the sensor data, and require no other sensor knowledge.

In this chapter, we discuss a scenario in which we have no external knowledge of the sensors capabilities (e.g. the signal-to-noise ratios), but instead only have the output data from the sensors. In this work, we assume that all the sensors are viewing the same scene. Since there is no external knowledge about the sensors other than the data itself, one can look at "agreement" and "overlap" in the sensors' data, since the sensors are all viewing the same scene. Herein, we provide a CM that estimates the conflict between the sensors, in order to understand how well each sensor's outputs agree with the outputs of other sensors viewing the same scene. Furthermore, using the CM for each sensor, one can fuse the sensor data in a more meaningful way. For instance, if we know that a sensor

has high conflict with the other sensors, we can assume that the other sensors are probably correct and this sensor is not, and we can de-emphasize the weighting of the conflicting sensor's outputs when we fuse sensor data together. It is noted that the proposed algorithm is designed to detect conflicts, meaning disagreement between sensors. It is quite another thing to know whether the majority is correct, or if the a single sensor that has a high CM in our method is actually correct. With no knowledge other than the sensor data, one can assume that the majority of non-conflicting sensors has the more correct data. Simply examining the conflicts with the proposed algorithm will not discriminate between these two cases.

By measuring the conflict among sensor data, and using this data, one can potentially facilitate better results in the next processing stage (e.g., sensor output fusion). In the fusion stage, lower weights will be assigned to sources with higher conflict with other sources; on the other hand, higher weights will be assigned to sources with lower CM.

There are numerous sources in the open literature which discuss methods to measure conflict, or in other words, the measure of distance, dissimilarity or divergence [4, 29, 69, 8]. On the other hand, the measure of similarity has been studied in many fields, such as fuzzy measures of agreement [34, 112, 108, 100] [37, 36] or similarity measures between probability density functions [8]. On some occasions, the measures of distance/dissimilarity/divergence are treated as one minus similarity measure or one over the similarity measure [18]. Herein, the method we propose to calculate conflict has a "one over similarity" relation with similarity measure. However, this relation could be extended to other negative associations.

4.2 Proposed Algorithm

In this section, the proposed CM is defined and discussed, as well as the algorithms to fuse data based on the proposed CM.

4.2.1 Conflict Measure

The basic idea behind the proposed CM algorithm is to give different weights to different overlapping regions based on how many times these sources' information overlaps in these regions. In a sense, the proposed measure is defining agreement by a combination of the interval sizes and how many sensors are reporting data in the intervals, which is a frequency of overlap approach. The nomenclature and mathematical description of the algorithm is as follows.

Let $X = \{x_1, ..., x_n\}$ be a set of sensor sources, for which we do not know a priori the quality of each source. Each source provides data, and subsets of the data can be grouped into interval-valued evidence ($\overline{h_i}$). For example, if sensor x_1 has output voltages v, where $2 \leq v \leq 3$, then the interval is $\overline{h_1} = [2, 3]$, and this is called interval-valued evidence. In this case, the interval-valued evidence was based on the minimum and maximum values in the sensor's outputs. This method, however, can lead to an over-inflated interval if there is significant noise or outliers in the data. In this work, we take the data, estimate the mean and standard deviation and set the interval-valued evidence to the mean plus or minus three standard deviations, which is more robust to outliers.

It is also noted that these intervals are not static since the sensor outputs can change over time. For instance, for a sensor taking data at regular time intervals, one choice is

let every N samples constitute a data subset. In the sequel, we will call this data subset a sub-interval. These sub-intervals can be disjoint or overlapping. In this work, we assume overlapping sub-intervals.

Define A_i as the set that contains all i-tuple combinations of the sources, where n is the total number of sources. For example, with sources x_1, x_2 and x_3, $A_1 = \{\{x_1\}, \{x_2\}, \{x_3\}\}$ are all the singletons, $A_2 = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_3\}\}$ are all combinations of pairs of sensors, and $A_3 = \{x_1, x_2, x_3\} = X$ are all triples. Note the sensor order doesn't matter, only the pairings.

Let $E = \{E_1, ..., E_{2n}\}$ be the ordered set $(E_i \leq E_{i+1})$ of all interval endpoints from our evidences. Let $\overline{P} = \{\overline{P_1}, ..., \overline{P_{2n-1}})$ be the set of intervals induced by E. Note that each interval requires two endpoints, so there are $2n - 1$ intervals since we have $2n$ endpoints. For example, $\overline{P_1} = [E_1, E_2]$. Let $O(\overline{P_k})$ be the number of sources with data in the interval $\overline{P_k}$. This is an integral part of our definition of the conflict measure, since more sensors agreeing on an interval means a lower conflict measure, and less sensor agreement means a higher conflict measure.

Let the conflict measure g^{CF} be defined as follows:

$$g^{CF}(A \in A_1) = 0, \forall A \in A_1 \tag{4.1}$$

$$g^{CF}(A \in A_i) = \frac{1}{E_{2n} - E_1} \sum_{k=1}^{2n-1} |\overline{P}_k| \left(\frac{i - O(\overline{P}_k)}{i} \right), i \in \{2, 3, ...n\},$$

Equation4.1 means that for each individual sensor there is no conflict, which is natural, since conflict between sensors really has no meaning for an individual sensor. Equation

4.2 shows that CF is defined here as the sum of sub-interval conflicts over all the sources' measurement range. Each sub-interval contributes to the conflict if some of the sources' measurement range does not cover such sub-interval. The conflict metric is a frequency of overlap approach based on the induced intervals. The overall length of the entire interval E is used as normalizing constant, since the sensor data can live on different intervals.

Once the conflict measures for each i-tuple have been calculated using Equation 4.1, 4.2, the overall conflict measure for each individual sensor, which is a weighted sum of relative to all the i-tuples are calculated as shown in algorithm 1, below. In algorithm 1, for simplicity, the scalar case is shown. The algorithm can easily be extended to vector data. In the vector case, each element, for instance, could be evaluated independently. For example, if the sensor provides 3D data, the algorithm could operate independently on the x, y, and z coordinates. Since the algorithm processes data in sub-intervals, the sub-interval length must be specified. The length of the evaluation sub-interval will be discussed below.

4.2.2 Fusion based on the Conflict Measure

To fuse the data, it is desired to weight the data such that the weight of a sensor diminishes as the CF increases. Let w_k be the weight of source k. One reasonable approach is to perform the weight calculation for source j is as in Equation4.2, 4.3:

$$w_j = \frac{\frac{1}{SumC_j}}{\sum_{i=1}^{n}(\frac{1}{SumC_i})} \tag{4.2}$$

$$SumC_j = \sum_{k=1}^{n} \frac{1}{k}(\sum_{A \in A_k} g^{CF}(A)) \tag{4.3}$$

Algorithm 1 Algorithm to Estimate the Conflict Measure

input: the number of sensors: n

input: $x_{i,j}$, the j-th input from the i-th sensor, $i = 1, 2, ..., n$

input: the number of data points in evaluation sub-interval: m

output: $g^{CF}(A \in A_i)$ for $i = 1, 2, ..., n$

start:

for each each sensor $i = 1, ..., n$ **do**

for each each sub-interval of data with starting index j **do**

Group the sensor data into a set of sub-interval data: $S_{i,j} = \{x_{i,j}, x_{i,j+1}, ..., x_{i,j+(m-1)}\}$

Based on the sub-interval data, estimate the mean value and standard deviation according to $u_{i,j} = \frac{1}{m}\sum_{k=0}^{m-1} x_{i,j+k}$ and $\sigma_{i,j} = \sqrt{\frac{1}{m}\sum_{k=0}^{m-1}(x_{i,j+k} - \sigma_i)^2}$

Set the interval-valued evidence $\overline{h_{i,j}} = [\mu_{i,j} - 3\sigma_{i,j}, \mu_{i,j} + 3\sigma_{i,j}]$

Calculate $g^{CF}(A \in A_i)$ for each element $A \in A_i$ according to Equation 4.1, 4.2

end for

end for

Equation4.3 is the weighted sum of the CF values across the lattice. For example, if there are n = 4 sensors, for sensor number 1,

$$SumC_1 = \tfrac{1}{1}g^{CF}(\{x_1\}) + \tfrac{1}{2}[g^{CF}(\{x_1, x_2\}) + g^{CF}(\{x_1, x_3\}) + g^{CF}(\{x_1, x_4\})] +$$

$$\tfrac{1}{3}[g^{CF}(\{x_1, x_2, x_3\}) + g^{CF}(\{x_1, x_3, x_4\}) + g^{CF}(\{x_2, x_3, x_4\})] + \tfrac{1}{4}g^{CF}(\{x_1, x_2, x_3, x_4\})$$

In this manner, the weights produced by Equation 4.2 will sum to one, and the increased values of CF for a given sensor will lower the weights for that sensor, and thus diminish the effects of sensors that are exhibiting high conflict. Once the weights are calculated, the fused data interval can be calculated as Equation 4.4 and Equation4.5:

$$FusedLeftEnd = LeftEnd_1 \times w_1 + LeftEnd_2 \times w_2 + LeftEnd_3 \times w_3 + LeftEnd_4 \times w_4 \tag{4.4}$$

$$FusedRightEnd = RightEnd_1 \times w_1 + RightEnd_2 \times w_2 + RightEnd_3 \times w_3 + RightEnd_4 \times w_4 \tag{4.5}$$

Finally, the data can be fused using a weighted sum based on the weights calculated in Equation 4.3 as follows, where the j-th data from sensor i is $x_{i,j}$. For a stereo camera sensor that returns 3D data, one can fuse the x, y and z data independently according to Algorithm 2.

$$FusedResult_j = \sum_{i=1}^{n} x_{i,j} w_i \tag{4.6}$$

Algorithm 2 Algorithm for conflict measure assessmentand fusion of 3D data .

input:$\{(x_{i,j}, y_{i,j}, z_{i,j})\}$ j-th tracking points from i-th sensor

input: the number of sensors: n

input: the number of data points in the conflict measure sub-interval: m

output: g^{CF}NM for x, y and z data

start:

Do these steps for each x value

for $i = 1, ..., n$ **do**

Calculate $g^{CF}(A \in A_i)$ for the x data according to Algorithm 1

end for

for $i = 1, ..., n$ **do**

Calculate the weights according to Equation 4.1, 4.2

end for

Calculate the fused interval using Equation 4.4, 4.5

Fuse the data according to the weights using Equation 4.6

Repeat for each y and z value

In Algorithm 2, the size of the conflict measure sub interval is provided as in input. This size will depend on many factors, and in the section below a reasonable choice of the sub-interval size based on the data is given, based on examination of several different sub-interval sizes.

4.3 Examples

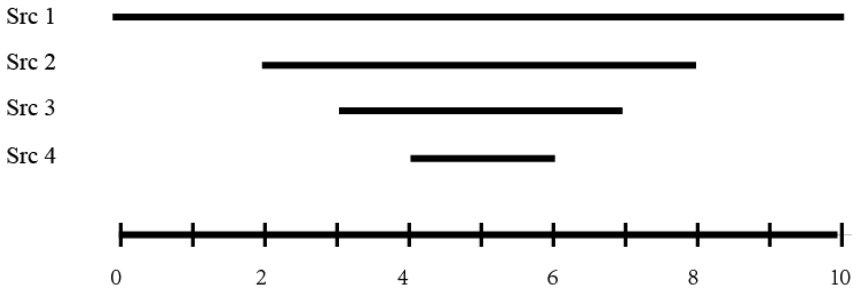
In this section, a simple numeric example is provided to illustrate the calculations involved. In addition, three stereo camera sensor 3D tracking examples are also provided.

4.3.1 Numeric Example

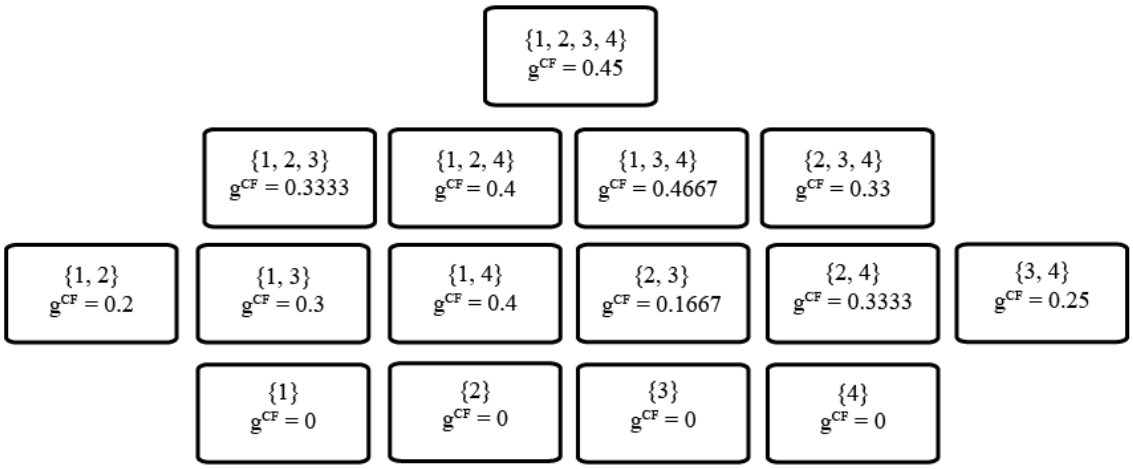
A synthetic example is given to illustrate the calculation of the CF. For the interested reader, more synthetic examples can be found in Chapter 3. For simplicity, we will assume we are processing only one sub-interval of sensor data. Suppose there are four interval-valued evidences, $\overline{h_1} = [0, 10], \overline{h_2} = [2, 8], \overline{h_3} = [3, 7]$, and $\overline{h_4} = [4, 6]$. The four evidences are shown in Figure 4.1(a), while the lattice of CF conflict measures is shown in Figure 4.1(b). For the calculation of $g^{CF}(\{x1, x2, x3\})$, as shown in Figure 4.2, intervals [0, 2] and [8, 10] have one source, sensor 1. Intervals [2, 3] and [7, 8] have two overlapping sources which are from sensors 1 and 2. Interval [3, 7] has all three overlapping sources. The CF is computed according to Equation 4.2 as follows:

$g^{CF}(\{x1, x2, x3\}) = (|0-2| \times \frac{3-1}{3} + |10-8| \times \frac{3-1}{3} + |2-3| \times \frac{3-2}{3} + |8-7| \times \frac{3-2}{3} + |3-7| \times \frac{3-3}{3}) \backslash (|10-0|) \approx 0.3333$

The results in Figure 4.1(b) show that CF is similar to what one might expect in terms of conflict. For example, the algorithm gives higher conflict measure to the set of sources x_1, x_2, x_4 than to the set of sources x_1, x_2, x_3. This is because there is more overlap to sources 1 and 2 from source 3 than from source 4.



(a) Four interval-valued sources.



(b) Lattice of CF values.

Figure 4.1

Synthetic Example for calculating CF.

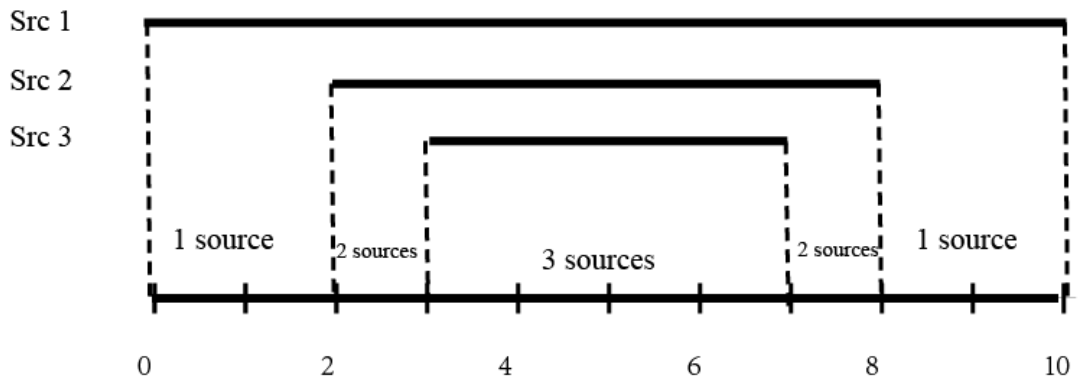


Figure 4.2

An example showing how to calculate $g^{CF}(\{x1, x2, x3\})$ of three sources.

To calculate the fusion weights of each source, we first calculate the sum of conflict measures for each source by adding the conflict measures that contain each particular source with a weight. Let be the overall sensor CM for source k, where $k = 1, 2, ..., n$. For example, if there were $n = 4$ sensors, then use Equation 4.3, we have $SumC_1 = 0.9625, SumC_2 = 0.8180, SumC_3 = 0.8486, SumC_4 = 1.0042$.

In the proposed method, it is desired that the weight assigned to each source have a negative correlation with the conflict measure. One can choose a function that negatively correlates the input and output, and then normalize it to make the sum of weights from all sources equal to one. In this example, we choose the simplest "one over" functions (e.g., $SumC^{-1}$). There are also other methods such as $SumC^{-n}$, where n could be any positive integer, and other possibilities.

In our example, the weight for sensor 1 is

$w_1 = \frac{\frac{1}{SumC_1}}{\sum_{i=1}^{n} \frac{1}{SumC_i}} = \frac{1/0.9625}{1/0.9625+1/0.8180+1/0.8486+1/1.00425} \approx 0.2342$

Similarly, $w_2 = 0.2756, w_3 = 0.2657, and w_4 = 0.2245.$

To fuse the three intervals together, one can use Equation 4.4 and 4.5 as follows:

$FusedLeftEnd = LeftEnd_1 \times w_1 + LeftEnd_2 \times w_2 + LeftEnd_3 \times w_3 + LeftEnd_4 \times w_4 = 0 \times 0.2342 + 2 \times 0.2756 + 3 \times 0.2657 + 4 \times 0.2245 = 2.2462$

The right end could also be calculated in a similar way, which evaluates to 7.7538. The fused result is show in Figure 4.3.

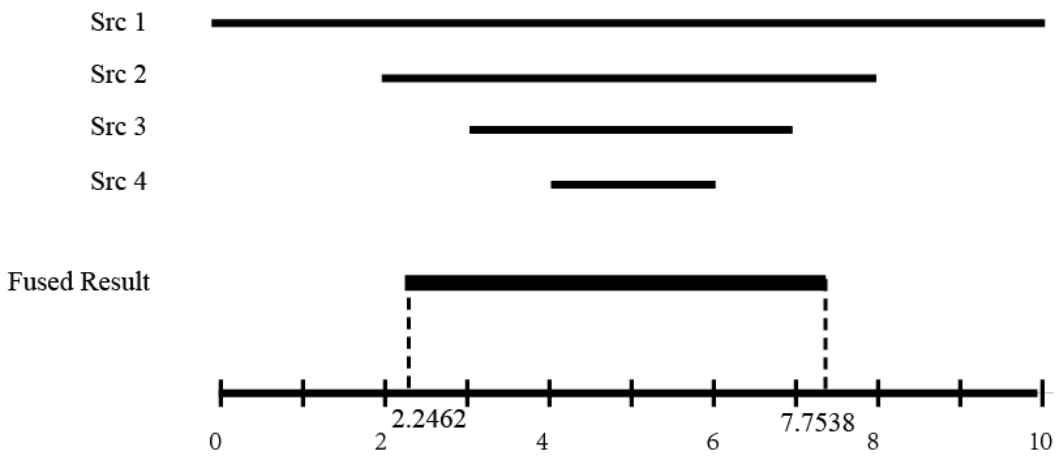


Figure 4.3

Fused result of four sources using the CF-based weights.

4.3.2 Stereo Camera Experimental Examples

The experimental setup is shown in Figure 4.4. Four pairs of stereo cameras provide four independent sources of tracking information in 3D space. In this experiment, the stereo cameras are all placed at nearby locations viewing the same object. The cameras

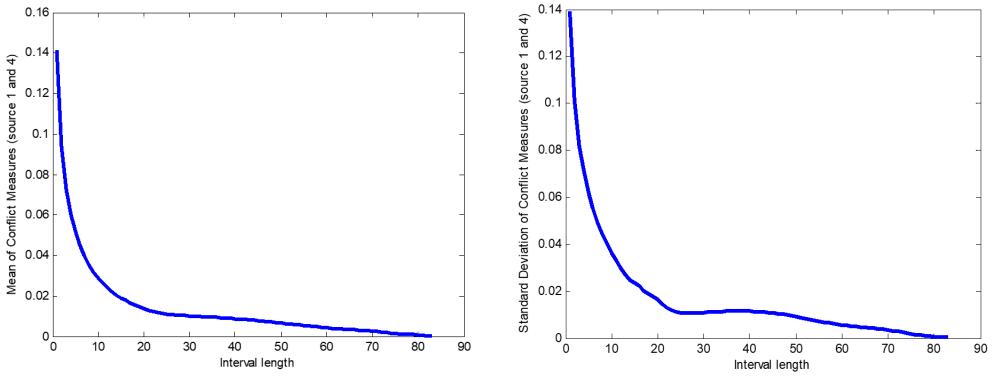
were calibrated using the checkerboard poster shown in the left of Figure 5. One stereo camera was chosen as the reference, and all other cameras images were adjusted to the measurement frame of the reference stereo camera [19]. An object was tracked in time by the cameras, and synthetic perturbations were added to stereo camera 2 in the form of additive impulse noise, additive DC offset (bias), and additive Gaussian noise. These examples were chosen to represent some common problems encountered in multi-sensor data. Impulse noise is common in sensors due to intermittent interference, a DC offset a sensor bias or registration error, and Gaussian noise represents a sensor output with low SNR.

Figure 4.4

Experimental setup showing the calibration board (to the left), and three of the four stereo camera pairs.

The calculation is similar to the synthetic synthetic example, and the only difference is that instead of using the function $SumC^{-1}, SumC^{-3}$ is employed, as it has a better result since it largely diminishes the effect of noise.

In this experiment, each stereo camera takes snapshots at a fixed time interval. It is assumed that each stereo camera is basically synchronized with respect to time (unsynchronized sources can be considered for future work). In the proposed method, a larger time sub-interval (which is an integer multiple of the sampling time) is chosen to evaluate the sensor measurements using the CF. In our experiments, it was noticed that with the change of the number of points in our sensor evaluation interval, the mean and standard deviation of the conflict measure also change. For example, Figure 4.5 shows the changing of mean and standard deviation of conflict measure using source 1 and source 4 data. After examination of the different sub-interval lengths, it was empirically discovered for our example case that around interval 10 all the curves change from a large slope down to a smaller slope down. In this case, the sub-interval size was chosen to be 10 times the sample size (that is, ten data points per interval) in the following three experiments. This value is in part due to the inherent 3D motion of the tracked object, as well as the dynamic nature of the conflict measure. Different scenarios may have different curves, and this effect will be studied in future work.



(a) Changing for the mean of conflict measure. (b) Changing for the standard deviation of conflict measure.

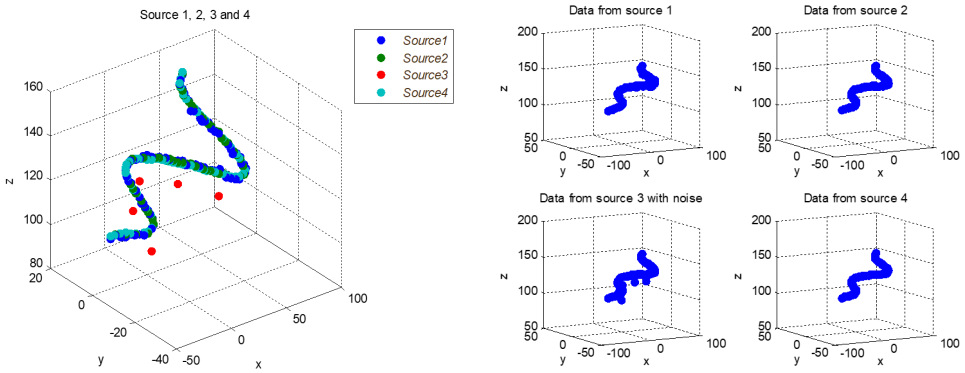
Figure 4.5

Mean and standard deviation of conflict measures (from source 1 and source 4) versus the overall interval length, which is an integer representing the number of samples.

The following three examples are based on different noise types which are artificially added to one of the sources in order for us to have a controlled amount of perturbation to the sensor outputs. The three types of noise added are impulse noise, a DC offset, and Gaussian noise. The impulsive noise represents some random interference. This type of noise can play havoc with sensor fusion algorithms due to the outlier nature of the corrupted data. The DC offset represent a misaligned source or a source that has some constant bias. Finally, the Gaussian noise represents a scenario with a poor sensor or a scenario where one sensor has low signal-to-noise ratios. In all of the experiments, sensor 1 and sensor 2 data were unaltered, while the noise was added to sensor three.

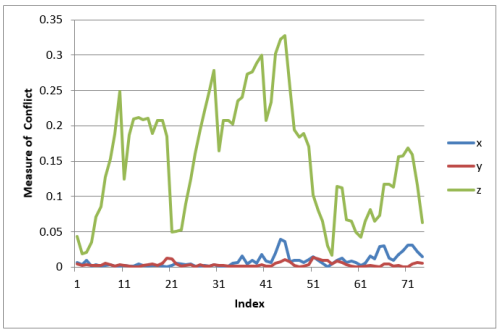
Example 5. Impulse noise

In this example, five constant impulsive noise of magnitude 100 are added to source 2's z values, and this noise source is treated as source 3, which is shown in Figure 4.6(a) and (b). The data is color coded as indicated in the legend for Figure 4.6(a). The result after fusion is shown in Figure 4.6(d). In the result, we can see that the impact of the impulsive noise is reduced.

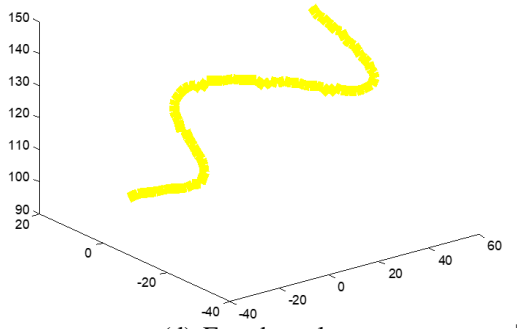


(a) Sensor outputs for the four sources, with added impulse noise.

(b) Sensor outputs for each source plotted individually.



(c) Conflict measure between sources 1 and 3 plotted versus the sub-interval length.



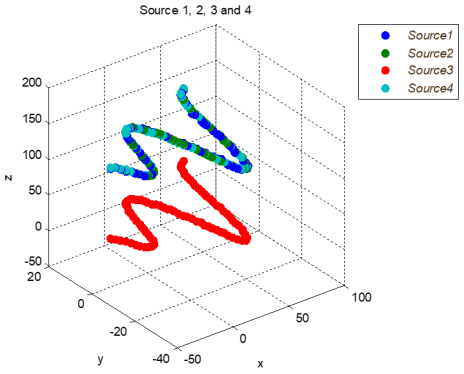
(d) Fused result.

Figure 4.6

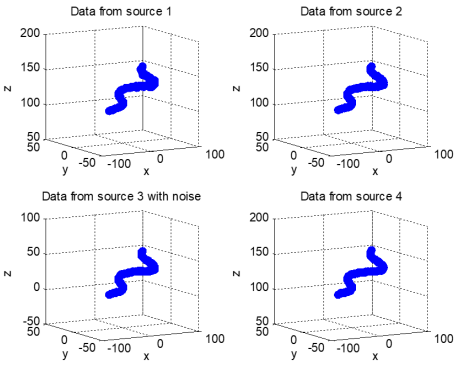
Impulse noise example.

Example 6. DC offset

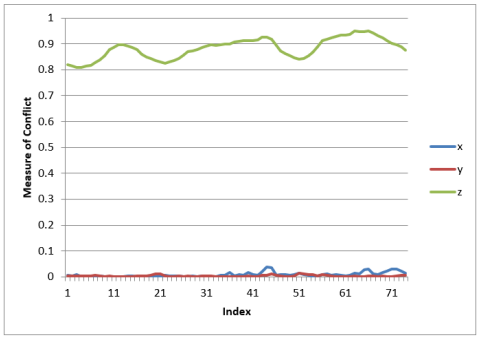
In this example, a DC or bias level (-1000) is added to source 2's z value, and this noise source is treated as source 3, as shown in Figure 4.7(a) and (b). This could be a common problem for multiple sensors where there is a misregistration error or some sort of bias in a distance estimation. Every z value of source 3 is lower than the other sources. The fused result in Figure 4.7(d) shows that the result largely reduces the impact of the source with DC offset.



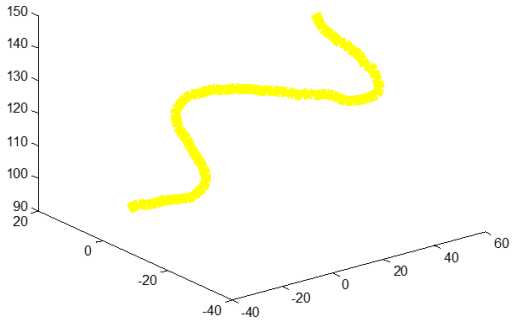
(a) Sensor outputs for the four sources, with added impulse noise.



(b) Sensor outputs for each source plotted individually.



(c) Conflict measure between sources 1 and 3 plotted versus the sub-interval length.



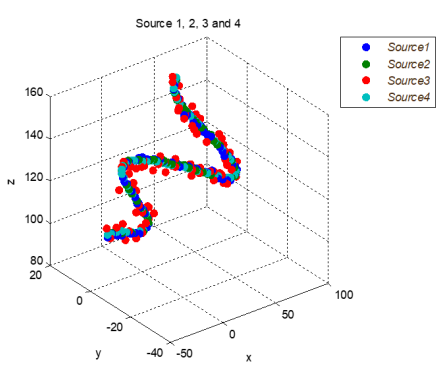
(d) Fused result.

Figure 4.7

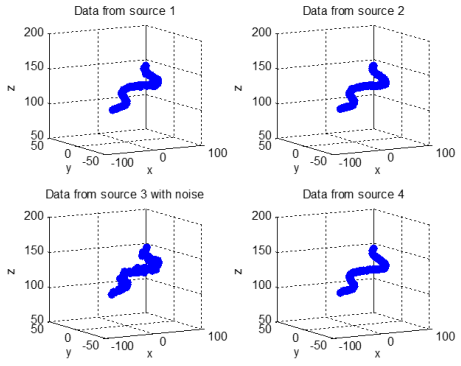
DC offset example.

Example 7. Gaussian noise

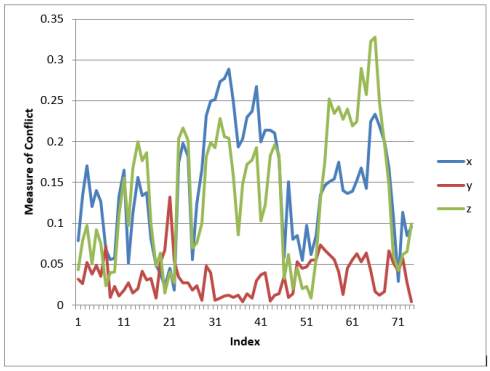
In this example, 30dB Gaussian noise is added to source 2's x, y and z value, and this noise source is treated as source 3, which makes the source not as smooth as the others, which are shown in Figure 4.8(a) and (b). This case mimics a sensor with a very poor SNR. This could be because of the sensor being inferior in quality to the other sensors, or due to some strong interference source. We can see that the fused result in Figure 4.8(d) has reduced the effect of the Gaussian noise.



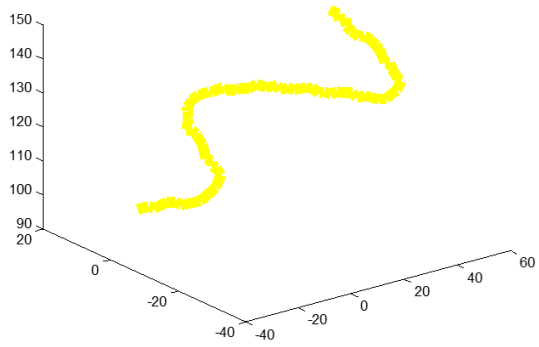
(a) Sensor outputs for the four sources, with added impulse noise.



(b) Sensor outputs for each source plotted individually.



(c) Conflict measure between sources 1 and 3 plotted versus the sub-interval length.



(d) Fused result.

Figure 4.8

Gaussian noise example.

In order to provide quantitative analysis of the results, source 1, source 2, and source 4 are the original measurements, while source 3 is generated by taking source 2 adding noise, we can suppose that source 1 is our benchmark (ground truth), and we can compare the average of absolute difference with source 1 using an average operator and our proposed method. For Impulse noise and DC offset, the noise is only added to z values; for Gaussian noise, the noise is added to x, y, and z values. As a result, we only compare the difference in z between source 1 and fused result in the first two scenarios (impulsive noise and DC offset), and compare the difference in x, y, and z between source 1 and fused result in the last scenario (Gaussian noise) as shown in Table 4.1. The equation for D is shown in Equation 4.7 as

$$\sum_{i=1}^{n} |A_i - B_i|/n \tag{4.7}$$

Table 4.1

Comparison between source 1 and the fused result using average operator or my method.

Noise type	Avg. D_x	Avg. D_y	Avg. D_z	Prop. D_x	Prop. D_y	Prop. D_z
Impulse noise	None	None	0.6508	None	None	0.5392
DC offset	None	None	25.0332	None	None	4.4719
Gaussian noise	0.5693	0.2653	0.6162	0.1424	0.1014	0.5542

From Table 4.1, we can see that our method provides performance improvements versus a simple weighted sum. In the impulsive noise case, the improvement is small since there are only five impulse noise signals added. In the DC offset case, the DC offset is

severe and the resulting improvement is significant. Note the x and y data in both the impulsive noise and DC offset cases were not degraded, so those entries in the table are not shown. In the Gaussian noise case, the improvements also show the diminished effects of the Gaussian noise.

4.3.3 Conclusions

In this chapter, a measure of conflict using interval-valued input is proposed. By evaluating the conflict measure, we can assess credibility of sources and fuse the information from different sources together. In the fused result, the effect of the sources with higher conflict measures is diminished, while the sources with lower conflict measure play a bigger role in the final fused results. Using the proposed algorithm to fuse 3D tracking information, the results we have are better than using average operator, which gives equal credibility to every source. This shows that conflict measure can be used as a way to calculate source credibility, and the fused result based on this credibility is better than an average operator.

CHAPTER 5

FUSION OF AN ENSEMBLE OF AUGMENTED IMAGE DETECTORS FOR ROBUST OBJECT DETECTION

5.1 Introduction

Object detection is an important and active area in machine learning research. In many instances, especially for Advanced Driver Assistance Systems (ADAS), it is important to accurately localize objects in real time. Due to the development of deep learning, the localization accuracy for object detection has largely been improved. However, in our experimenting with deep learning for object detection (such as detecting pedestrians), we noticed that even with a well-trained network with image augmentation (such as translation, rotation, stretching) during training, changes in the image characteristics such as brightness or contrast can still have a large effect on the detection result. Based on these observations, we hypothesize that presenting a deep network with multiple augmented images during *testing* and fusing the results could result in a more robust detection system. Herein, we develop a system utilizing image augmentation combined with Axis-Aligned Bounding Box Fuzzy Integral (AABBFI)-based fusion to enhance the detection results. The fusion method applied in this paper originates from the field of computational intelligence. This method analyses the agreement among object detection outputs and fuses them dynamically. We choose not to use a supervised learning-based method (learn the fusion

from training data), because the fusion method in this scenario cannot be learned in general, as the "optimal" fusion result changes from case to case. Specifically, even with the same inputs are to be fused, the expected fusion result is different when the environmental surroundings and/or detection object type changes. Instead, the method we apply in this paper automatically evaluates the characteristics among inputs for each case and fuses the detection results dynamically.

The primary research task we were investigating for this paper is developing a robust detection sub-system for ADAS. During road construction, street repairs and accidents, often, traffic lanes are quarantined with barrels or cones. Furthermore, ADAS detect pedestrians to estimate their locations and help with collision avoidance. In our specific industrial project, we also need to detect boxes. This motivated the study of cone, pedestrian and box detection herein, and a dataset consisting of traffic cones, pedestrians and boxes has been collected and analyzed. In the experimental results, we also include results using the PASCAL VOC (Visual Object Classes) dataset [22] for generalization purposes.

This paper shows that the proposed method gives better detection results than both the original non-fusion results and the average/median fusion results. Specifically, the contributions of this work are:

- A detection fusion system is proposed, which uses online image augmentation *before the detection stage* (i.e., during testing) and fusion after to increase detection accuracy and robustness.
- The proposed fusion method is a computationally-intelligent approach, based on the dynamic analysis of agreement among inputs for each case.
- The proposed system produces more accurate detection results in real time; therefore, it helps with developing an accurate and fast object detection sub-system for robust ADAS.

- The Choquet Integral (ChI) is extended from a one-dimensional interval to a two-dimensional axis-aligned bounding box (AABB).

5.2 Background

5.2.1 Object Detection using Deep Learning in ADAS

In recent years, deep learning has become the focus of much research. The beginning of the surge in deep learning was the winning of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [17] by AlexNet [50] developed by Alex Krizhevsky et al. Since then, deep learning has been integrated in many systems with outstanding performance, as shown in [107, 94, 96, 39].

Due to the development of deep learning, the accuracy of object detection in images has largely been improved. Examples of these object detectors includes: YOLO [85, 86], Faster R-CNN [87], Multibox [97], Region-based Fully Convolutional Networks (R-FCN) [13] and the Single Shot multibox Detector (SSD) [61].

Furthermore, there are various methods focusing on improving localization accuracy. One approach utilizes a probabilistic method. One example of this approach is called an object Localization Network (LocNet) [28], which relies on assigning conditional probabilities to rows and columns of the bounding box. LocNet exceeds the performance of Fast R-CNN [30], but it has been surpassed by Faster R-CNN [87]. A second approach focuses on improvement of Non-Maximum Suppression (NMS) [25]. One example is the so–called Gossip Net (Gnet) [41], which incorporates NMS into the detection network, but requires a large number of training data. In another example, the authors gave slight modification to NMS, but the improvement typically only happens in some specific cases [5].

A third type of approach uses neural networks to refine the localization. One example is RefineNet [59], which iteratively pools features from previous predictions. However, its performance using deeper networks is yet to be studied. Our approach is different from these three types. We use a computationally-intelligent method to fuse different detection results obtained from augmented images.

For ADAS, two of the most important metrics for the object detection sub-system are accuracy and speed. When using ADAS to help a driver in the process of driving, we first need the system to accurately detect objects near or far, big or small in various environmental conditions. Misdetection or false detection could possibly lead to disasters. Second, the detection needs to be in real time.

In our application for an industrial collision avoidance system, the maximum speed of the vehicle is about 5 m per second, which is roughly 18 km/h. To guarantee a safe stop for collision avoidance, the maximum computational time is 0.2 s, so real time in our application is defined as having a detection rate of 5 Hz or above. Other systems with different parameters will have different requirements.

5.2.2 Image Augmentation

In many deep learning systems, there is a large number of parameters to be estimated, and thus, a large amount of training data is required. Often, augmentation techniques such as image translation, rotation, stretching, shearing, rescaling, etc., are utilized to provide augmented imagery for training, and these image augmentation techniques during training often help the system be more robust. After performing a thorough literature search,

we found that there were rarely any previously published systems that utilized multiple augmentations and fusion for processing test images. One system is Howard's [42] entry for ILSVRC [17] in 2013, in which he modified the winning 2012 entry [50] by adding augmentations in both training and testing. The training augmentations were (1) extended pixel crops, (2) horizontal flips and (3) manipulating the contrast, brightness and color. For testing, a combination of 5 translations, 2 flips, 3 scales and 3 views yielded 90 enhanced images. A greedy algorithm then down-selected this to the top ten predictions, which were combined to form the final result. However, this augmentation employed during testing [42] yielded an increased classification accuracy but did not improve detection localization accuracy.

5.2.3 Model Ensembles

In order to improve performance during testing and add some robustness to the system, one idea is to utilize ensembles of multiple models. In this approach, one would train multiple models and average their predictions during the test time. One example is in a YOLO paper [85], where the authors combined Faster R-CNN and YOLO together and obtained some performance improvement. In [45], rather than training separate models independently, the authors kept multiple snapshots of the model during training. At test time, they averaged the predictions of these multiple snapshots.

Malisiewicz et al. [66] trained a linear SVM for each class in the training set. The main disadvantage is each class requires millions of negative examples to train. Szegedy et al. [97] demonstrated that proposal generation can be learned from data (vs. hand-crafted)

and utilized an ensemble approach to aggregate detections. Their solution was limited to one object per grid entry with a course grid, vs. YOLO, which has a finer grid and allows multiple objects to be detected per grid location. Maree et al. [67] utilized ensembles of extremely randomized decision trees and randomly extracted subwindows for object detection. The tree sizes were very large, and this method may not be appropriate for a real-time system. There are also unsupervised methods such as applying an ensemble clustering to extract ROIs from low depth of field images by Rafiee et al. [83]. Performance improvements could be obtained if this were a supervised method.

There are two major differences between the model ensemble approach and our proposed method. First, instead of using multiple models, we use one model, and we change the inputs of the model instead of the model itself. Second, instead of averaging all predictions, we propose a computationally-intelligent method to fuse all predictions together.

5.3 Proposed System

5.3.1 Overview

For the proposed system, during the in-line (testing) phase, the input goes through three stages. First, the input is augmented to produce several variations, so we can have augmented inputs for future stages. Second, a detector is applied to obtain AABBs and related labels in each augmented image. Practically, this would be implemented by applying multiple detectors in parallel, one for each of the augmented images. This system can handle multiple objects in the input image. The largest number of detected objects in the augmented inputs is chosen as the number of objects (S) in the input. For example, suppose three augmented images provide 3, 2 and 2 detections; we then choose the con-

servative approach and set $S = 3$. All AABBs are grouped for each object using k-means clustering [63], where k equals S. Third, for each object, a certain number of AABBs with the top T confidence scores in augmented inputs is selected, where T can be any integer that is equal to or smaller than the number of augmented inputs (M). Then, the AABBFI fusion method is used to fuse the T AABBs to obtain one AABB for each object in the input. In the ADAS examples in Section 5.4, we use three AABBs for tractability, but more AABBs can be fused using the method proposed in [47].

In summary, the system produces variations of each input, detects objects in each variation and fuses the top T results for each object, with the expectation of getting a more accurate AABB for each object in the input. The overview of the proposed system is shown in Figure 5.1, and Algorithm 3 is a formal description of the proposed system.

In the following paragraphs, each part of the proposed system, including augmentation, detection and fusion, is discussed in detail.

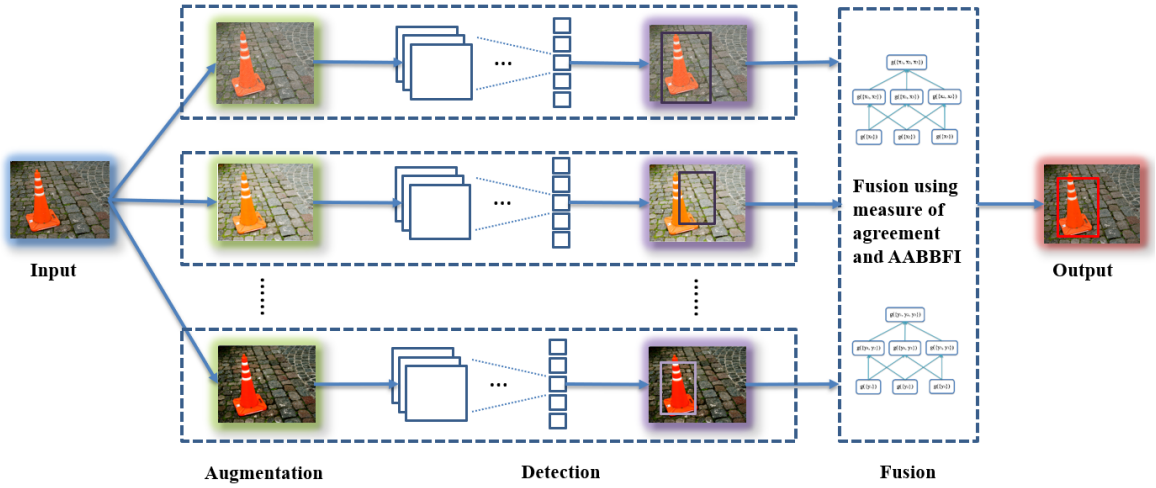


Figure 5.1

Proposed system for detection fusion.

Algorithm 3 Algorithm for the proposed detection fusion system.

input: input: $image\, I$

output: fused AABBs for S objects: $\{\hat{B}_1, \hat{B}_2, ..., \hat{B}_S\}$

global: augmentation methods: $\{P_1, P_2, ..., P_M\}$

global: augmented inputs: $\{\hat{I}_1, \hat{I}_2, ..., \hat{I}_M\}$

global: number of objects in input: S

global: AABBs for all objects: $\{B_1, B_2, ..., B_X\}$

global: labels for all objects: $\{l_1, l_2, ..., l_X\}$

global: AABBs for each object: $\{B_1, B_2, ..., B_N\}$, N may vary

global: labels for each object: $\{l_1, l_2, ..., l_N\}$, N may vary

global: Number of AABBs to be fused: T

for each augmentation method in $\{P_1, P_2, ..., P_M\}$ **do**

Use the augmentation method on I to obtain augmented inputs $\{\hat{I}_1, \hat{I}_2, ..., \hat{I}_M\}$.

end for

Use the detector to get $\{B_1, B_2, ..., B_X\}$ and $\{l_1, l_2, ..., l_X\}$ for all objects of all augmented inputs.

Choose the largest number of detection in each variation as S.

Group the detection into S object groups using k-means clustering [63], where k equals S.

for k from 1 to S **do**

Choose the grouped detection results $\{B_1, B_2, ..., B_N\}$ and $\{l_1, l_2, ..., l_N\}$ for each object, where these N detection results from N augmented images correspond to the same object. Here, N may vary.

Choose the majority in $\{l_1, l_2, ..., l_N\}$ as the object label.

if $N \geq 3$ **then**

Fuse T AABBs with the top T confidence scores using AABBFI to get $\hat{B}_k$

else if $N = 2$ **then**.

Average the AABBs to get $\hat{B}_k$.

else if $N = 1$ **then**

Use only this detection AABB as $\hat{B}_k$.

else if $N = 0$ **then**

$\hat{B}_k = None.$

end if

end for

return $\{\hat{B}_1, \hat{B}_2, ..., \hat{B}_S\}$.

5.3.2 Augmentation

For augmentation of each input, the goal is to produce a range of inputs. The "optimal" augmentation cannot be determined algorithmically, and it varies depending on the type of objects detected, the image background, etc. Instead, a range of images of varying quality is generated and presented to the detector. The augmentation methods used herein include changing brightness, contrast, edge enhancement, global histogram equalization, Gaussian blurring and adding independent and identically distributed (IID) Gaussian noise to simulate different scenarios. Although there are other augmentation methods, in this initial work, we choose to focus on basic operations, which we have already shown lead to success. Future investigations can study other more complicated local and global operators.

In order to see whether all these basic augmentation methods work for the final result, we choose 17 of them to test on 572 images of traffic cones and summarize the choice of each augmentation method. During testing, each augmentation method that is chosen by the proposed algorithm as the top three choices is tallied. Detection algorithms usually output confidence scores (as in YOLO [85, 86]) or objectness scores (as in Faster R-CNN [87]) for each detected object. Herein, we use these scores to determine the top choices.

We run our system and count how many times each enhancement method is chosen. The results are as shown in Figure 5.2. The number of times selected for each augmentation method is shown on the x-axis. The different augmentation methods are listed on the y-axis, including the parameters for each method. For example, "brightness with factor 0.25" means the image is darker than the original image, while "brightness with factor 2" is two times brighter than the original image.

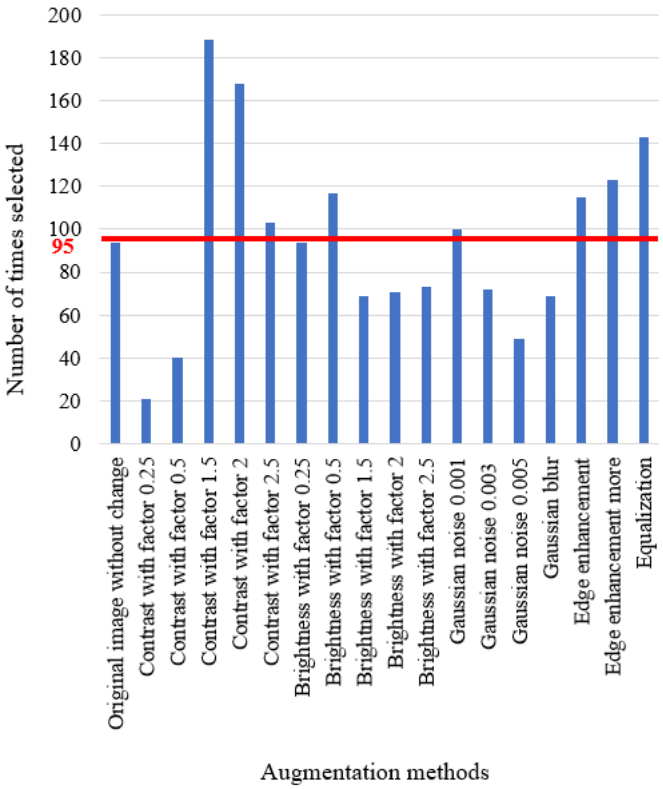


Figure 5.2

Augmentation analysis.

From Figure 5.2, we can see that each augmentation method is chosen in certain scenarios as one of the top three choices. For each method, if it is chosen equally, the average number of chosen times would be around 95, as shown in the figure as a red line. In Figure 5.2, the highest number of chosen times is 188, when using contrast with factor 1.5. The lowest chosen times is 21, when using contrast with factor 0.25. However, all these numbers fluctuate around the average 95, which means that each augmentation method has a certain influence on the final result, which cannot be ignored. This figure also shows that

there is really no overall "magic-bullet" method that universally helps for all images and the detection of all objects.

As the computational time scales linearly with the number of augmentation methods, we want to evaluate how the performance of the fusion system changes as the number of augmentation methods increases. In order to do this, the methods shown in Figure 5.2 are sorted in descending order based on the number of times each was selected during training. The original image without any augmentation is used as a starting point. For M augmented images, the M methods that were selected the most often are utilized. For instance, when using $M = 3$ augmented images, the contrast with factor 1.5 and contrast with factor 2 methods are chosen. Together with the original image, we can have three input images. The results are shown in Figure 5.3. From this figure, we can see that the performance reaches the highest point when we choose all 18 augmentation methods. However, the results also show fluctuations between one and 11 methods. It is important to note that when the number of augmentation methods is three, the performance reaches a local maximum point. This means that although the general trend line is going up, when computational resources are limited or there is higher speed requirement, we can choose three methods instead of 18 and still get improved performance. In Figure 5.3, the number of augmented methods is varied, and the IoU fusion result is calculated. The fusion really only occurs when $M \geq 3$ methods are used. The dashed vertical lines show the limits of what was examined in this paper, the $M = 3$ to $M = 18$ cases. The dashed blue line shows a trend line fitted to the data. The red dashed lines show the minimum (3) and maximum

(18) number of augmentation methods investigated. The solid blue line shows the data trend (the data points are blue dots).

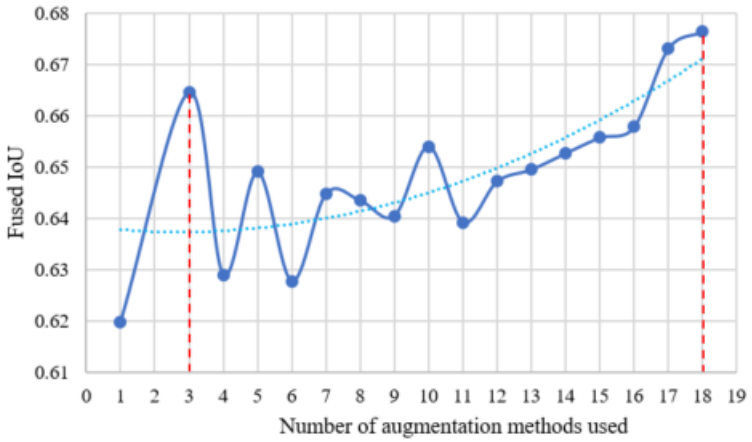


Figure 5.3

Changes of performance with the increase of the number of augmentation methods.

5.3.3 Detection

There are basically two types of detection (localization) methods using deep learning. One type is based on region proposals, and one example is Faster R-CNN [87]. For Faster R-CNN, the input image first runs through some convolutional layers to obtain a feature map representing the entire image, then there is a separate region proposal network, which works on top of the convolutional feature map and predicts its region proposals. Once we have these predicted region proposals, we pass them up to the rest of the network to get a classification of these regions. Therefore, there are two parts in the network; one is for proposing regions, and the other is for classification.

The other type of detection method detects without proposing possible regions. Instead, it localizes and classifies objects using one giant convolutional network. One typical example for this type of detector is called YOLO [85, 86]. Given an input image, YOLO first divides it into some coarse grids. Within each of these grid cells, YOLO assigns some base bounding boxes. For each of these base bounding boxes, YOLO predicts an offset off the base bounding box to predict what is the true location of the object off this base bounding box, a confidence score for the location and classification scores. To summarize, YOLO uses a single neural network for simultaneously predicting the location of the object, the confidence score for the location and classification scores.

Compared with methods based on region proposals, YOLO is considered to be real time, but it also has relatively high detection accuracy. One version of YOLO training on the PASCAL VOC dataset [22] can achieve 40 Frames Per Second (FPS) and 78.6 mean Average Precision (mAP), while with the same training dataset, Faster R-CNN obtains 70 mAP and 0.5 FPS. In the YOLO paper [85], the authors admitted that they struggled to localize small objects, which can be mitigated by training special detectors for small objects, as shown in [44].

One thing we observe from the experiment is that YOLO is more sensitive to input changes than Faster R-CNN, which means that there would be more variety in YOLO's outputs. This variety is a good fit for our proposed AABBFI method to produce a more accurate result. Therefore, YOLOv2 [86] is chosen in our proposed system to achieve real-time accurate detection for robust ADAS.

5.3.4 Fusion

For the fusion part, the FM of agreement as defined in Section 2.1.2 is used. Herein, the AABB Fuzzy Integral (AABBFI) is proposed to fuse AABBs.

In this paper, evidence is represented as an AABB. In other words, evidence is an AABB indicating where the network delineated the detected object's boundaries. For example, an AABB can be represented by the coordinates of upper-left and bottom-right vertices, such as $[1, 2, 3, 4]$, which means that the coordinates for the upper-left vertex are $[1, 2]$, and the coordinates for the bottom-right vertex are $[3, 4]$. However, the FI have only been extended to one-dimensional interval-valued information. Herein, we extended it to two dimensions and specifically AABBs. Note, the advantages of an AABB are that (i) it is a convex and normal set and (ii) it can easily be decomposed without loss into two separate intervals, one on the row and one on the column axis. Hereby, the AABBFI is defined as follows:

Definition 5.3.1 (AABB fuzzy integral) *Let $\overline{\overline{h_i}}$ be the AABB evidence from the i-th input. Let $\overline{\overline{h_{x,i}}} \subseteq I$ be the interval-valued evidence for the x-axis of the i-th AABB from the i-th input. Furthermore, let $[\overline{\overline{h_{x,i}}}]^-$ and $[\overline{\overline{h_{x,i}}}]^+$ be the left and right endpoints of interval $\overline{\overline{h_{x,i}}}$. In the same way, let $\overline{\overline{h_{y,i}}} \subseteq I$ be the interval-valued evidence from AABB's y-axis from the i-th input, and let $[\overline{\overline{h_{y,i}}}]^-$ and $[\overline{\overline{h_{y,i}}}]^+$ be the top and bottom endpoints of interval $\overline{\overline{h_{y,i}}}$. Herein, the membership at each location of the AABB is assumed to be one. The AABBFI can be computed as follows,*

$$\int \overline{\overline{h}} \circ g = [\int [\overline{\overline{h_x}}]^- \circ g, \int [\overline{\overline{h_y}}]^- \circ g, \int [\overline{\overline{h_x}}]^+ \circ g, \int [\overline{\overline{h_y}}]^+ \circ g]. \tag{5.1}$$

Because an AABB is convex and the heights of all of our sets are one, we can disregard the third dimension (the "value" at each location in the AABB). These AABBs are assumed herein to be binary sets. Furthermore, the axis-aligned convexity makes proving Equation (5.1) trivial because it is easily decomposed; meaning each AABB can clearly be represented by the union of two closed intervals, and one can compute that as two separate individual interval-valued integrals and union the result (verifiable by the extension principle [16]).

5.4 Examples and Discussion

In order to explain and validate the AABBFI fusion method, three synthetic examples are given. Then real-world ADAS examples are analyzed to reinforce the proposed system's efficacy.

For comparison with the proposed fusion method, two expected value operations, average and median, are chosen. They are two of the most commonly-used methods for summarizing multiple inputs into one output, and median operation is supposed to be robust to noise.

5.4.1 Synthetic Examples

The following are three synthetic examples of different scenarios, where the first example has three similar AABBs; the second example has three AABBs with one outlier; and third example has three AABBs with one extreme outlier. These three different synthetic

examples show how this fusion process works and give us some intuition about why the proposed AABBFI method works better than average and median operations.

Example 8. Three similar AABBs:

In the first example, there are three AABBs to be fused. The reason these AABBs are chosen is that they have the same size, and they overlap, which means that they are similar. The top and bottom vertices' coordinates ($[x_{top}, y_{top}, x_{bottom}, y_{bottom}]$) are used to represent each AABB. The coordinates of the three AABBs are $[1, 1, 4, 6]$, $[2, 2, 5, 7]$, $[3, 3, 6, 8]$, as shown in Figure 5.4. Note that the x and y axes units are arbitrary units of measure. The lattice of the FM of agreement is shown in Figure 5.5. The result using the proposed AABBFI method is shown in Figure 5.4 with the red dashed line, while the average result is shown with the yellow dashed line, and the median result is shown with the purple dashed line for comparison. The coordinates of the AABB obtained from the proposed AABBFI method are $[1.44, 1.42, 4.44, 6.42]$; the average AABB's coordinates are $[2, 2, 5, 7]$; and the median AABB's coordinates are $[2, 2, 5, 7]$.

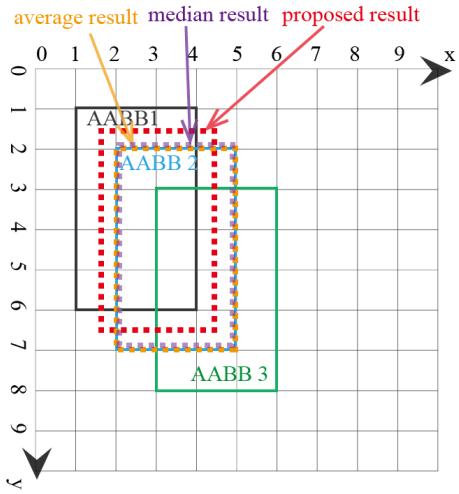
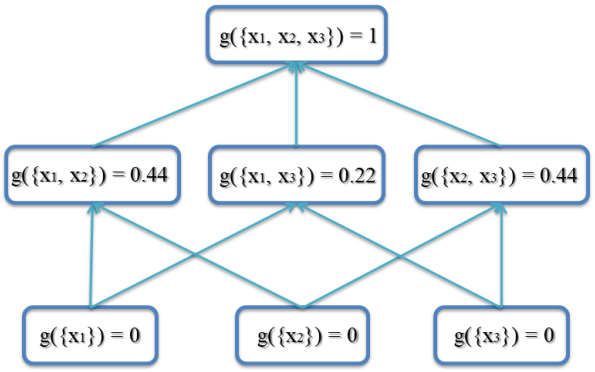
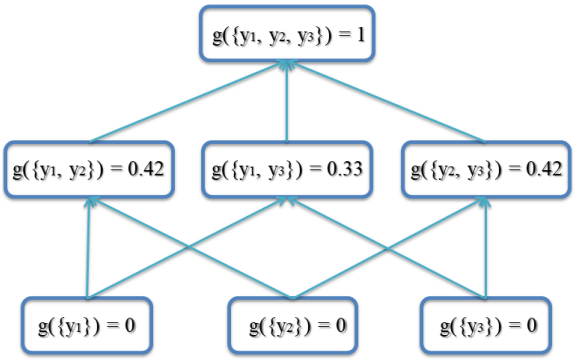


Figure 5.4

First example of fusing three similar AABBs.



(a) Lattice of FM of agreement on x axis for the first example.



(b) Lattice of FM of agreement on y axis for the first example.

Figure 5.5

Lattice of FM of agreement for the first example of three similar AABBs.

For the calculation of the FM of agreement, take the calculation of $g(\{x_1, x_2\})$ as an example. For the x-axis, the three intervals to be fused are $x_1 = [1, 4], x_2 = [2, 5]$ and $x_3 = [3, 6]$. From Equation (2.1b), we have:

$$\tilde{g}^{AG}(\{x_1, x_2\}) = \Big|[1, 4] \cap [2, 5]\Big| \times \frac{2}{3} = \Big|[2, 4]\Big| \times \frac{2}{3} = 2 \times \frac{2}{3} = \frac{4}{3}$$

$$\begin{aligned}\tilde{g}^{AG}(\{x_1,x_2,x_3\}) &= \Big|([1,4]\cap[2,5])\cup([1,4]\cap[3,6])\cup([2,5]\cap[3,6])\Big| \\ &\quad\times\frac{2}{3}+\Big|[1,4]\cap[2,5]\cap[3,6]\Big|\times\frac{3}{3} \\ &= \Big|[2,4]\cup[3,4]\cup[3,5]\Big|\times\frac{2}{3}+\Big|[3,4]\Big|\times\frac{3}{3} \\ &= \Big|[2,5]\Big|\times\frac{2}{3}+1\times\frac{3}{3}=3\times\frac{2}{3}+1=3\end{aligned}$$

From Equation (2.1c), the calculation is:

$$g^{AG}(\{x_1,x_2\})=\frac{\frac{4}{3}}{3}=\frac{4}{9}\approx 0.44.$$

This value is shown in the shaded box in Figure 5.5a.

Next, we show how to calculate the x_{top} coordinate using the proposed method. The x_{top} coordinates from the three AABBs are 1, 2 and 3, which means that the permutation sorting gives $[\overline{\overline{h}}(x_{\pi(1)})]^- = 3, [\overline{\overline{h}}(x_{\pi(2)})]^- = 2, [\overline{\overline{h}}(x_{\pi(3)})]^- = 1$. By using the FM in Figure 5.5a, we can calculate the x_{top} using Equations (2.2) and (5.1) as follows:

$$\begin{aligned}x_{top}=\textstyle\int[\overline{\overline{h_x}}]^-\circ g \quad &= 3\times[g(\{x_3\})-0]+2\times[g(\{x_2,x_3\})-g(\{x_3\})] \\ &\quad +1\times[g(\{x_1,x_2,x_3\})-g(\{x_2,x_3\})] \\ &= 3\times(0-0)+2\times(0.44-0)+1\times(1-0.44)=1.44.\end{aligned}$$

In this example, three AABBs are similar, with the same size and overlapping with each other. Therefore, we expect the fused AABB to be influenced by all three AABBs, both in shape and position. From the result, we can see that the majority of the proposed AABB is the overlapping area of the three AABBs, and it has the same size, just like our expectation. On the other hand, the average AABB and median AABB are exactly the

same as AABB 2, which may be or may not be influenced by AABB 1 and AABB 3. In the case of fusing three similar AABBs, the proposed method gives a result that fits our natural expectations.

Example 9. Three AABBs with one outlier:

In the second example, there are three AABBs to be fused, with one possible outlier being AABB 3, as AABB 3 has no overlapping area with AABBs 1 and 2. The three AABBs are $[1, 1, 4, 6]$, $[2, 2, 5, 7]$, $[7, 4, 10, 9]$, as shown in Figure 5.6. The lattice of the FM of agreement is shown in Figure 5.7. The proposed AABBFI result is shown in Figure 5.6 with the red dashed line, with the average result with the yellow dashed line and median result with the purple dashed line. The proposed AABB's coordinates are $[1, 1.38, 4, 6.38]$; the average AABB's coordinates are $[3.33, 2.33, 6.33, 7.33]$; and the median AABB's coordinates are $[2, 2, 5, 7]$.

In this example, AABB 3 has no overlap with the other two AABBs, and it seems like an outlier even though it is the same size. We expect that the fused result should have more agreement with the first two AABBs. From the result, it is shown that the AABB from the proposed AABBFI method is most similar to AABB 1 and influenced by AABB 2 without any overlap with possible outlier AABB 3. On the other hand, if averaging the three AABBs, the result is shifted towards AABB 3, which means that the average result is influenced heavily by AABB 3. The median result is again the same as AABB 2, without any influence from either AABB 1 or AABB 3. The two results by average and median operations do not fit our expectation, while the result produced by the AABBFI method fits

our expectation that the fused result is largely influenced by AABB 1 and AABB 2, with little influence from AABB 3.

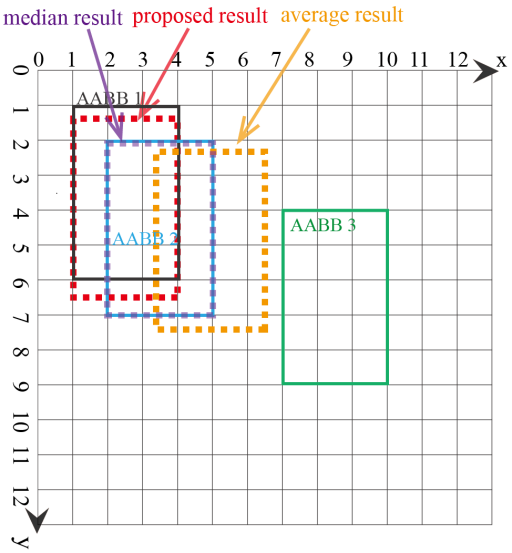
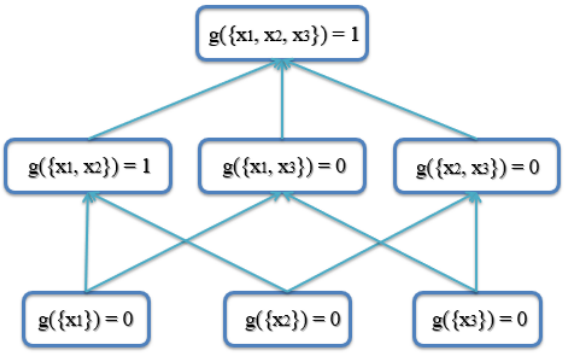
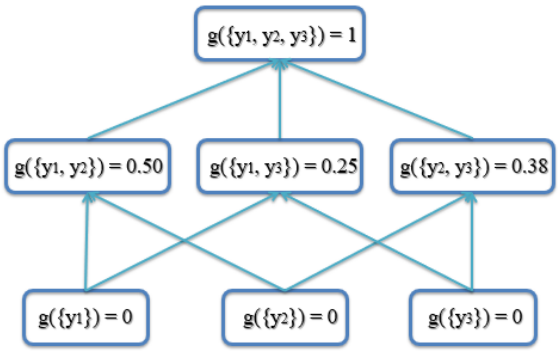


Figure 5.6

Second example of fusing three AABBs with one outlier.



(a) Lattice of FM of agreement on x axis for the second example.



(b) Lattice of FM of agreement on y axis for the second example.

Figure 5.7

Lattice of FM of agreement for the second example of three AABBs with one outlier.

Example 10. Three AABBs with one extreme outlier:

In the third example, there are three AABBs to be fused with one obvious outlier AABB 3, which has a much smaller size than the other two and has no overlapping area. The three AABBs are $[1, 1, 4, 6]$, $[2, 2, 5, 7]$, $[7, 8, 8, 9]$, as shown in Figure 5.8. The lattice of the FM of agreement is shown in Figure 5.9. The result from the proposed AABBFI method is shown in Figure 5.8 with the red dashed line. The average result is shown with the yellow

dashed line and the median result with the purple dashed line. The proposed AABB's coordinates are $[1, 1, 4, 6]$; the average AABB's coordinates are $[3.33, 3.66, 5.67, 7.33]$; and the median AABB's coordinates are $[2, 2, 5, 7]$.

In this example, AABB 3 looks like an obvious outlier and should not influence the final fused result at all. From the fused result using the AABBFI method, we can see that the fused AABB is the same as AABB 1, without any influence from AABB 3, which is the same as our expectation. On the other hand, the average AABB is largely influenced by AABB 3. The average AABB not only shifts towards AABB 3, but also shrinks in size, which is different from our expectation. Once again, the median result is the same as AABB 2. Comparing the three fusion results, AABB by the proposed AABBFI fits our expectation most.

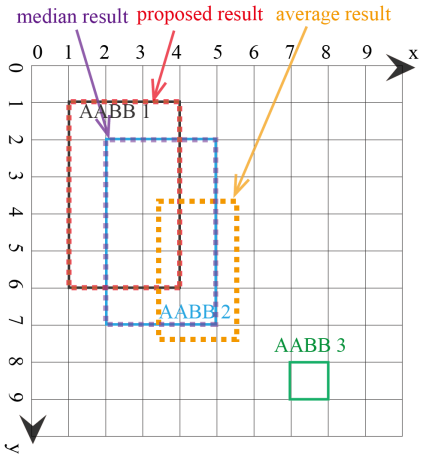
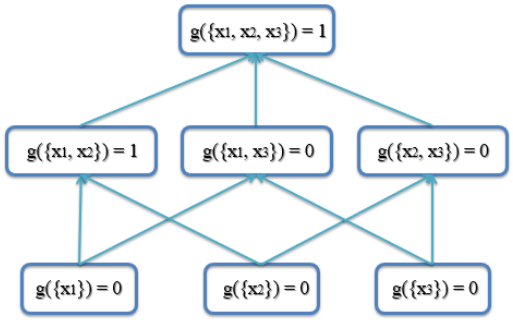
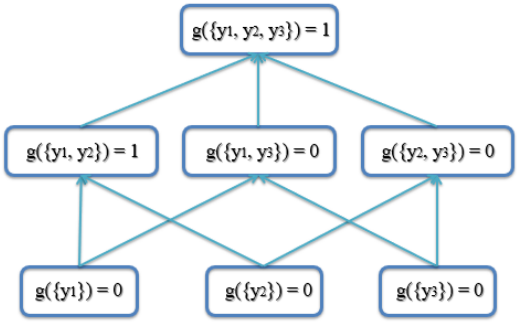


Figure 5.8

Third example of fusing three AABBs with one extreme outlier.



(a) Lattice of FM of agreement on x axis for the third example.



(b) Lattice of FM of agreement on y axis for the third example.

Figure 5.9

Lattice of FM of agreement for the third example of three AABBs with one extreme outlier.

5.4.2 ADAS Examples

In the ADAS examples, we focus on the accurate detection of three types of objects: cones, pedestrians and boxes. The proposed system is also tested on all of the PASCAL VOC 2007 testing dataset, and mAP is calculated to show general improvement on detection [22, 23].

One metric for measuring the accuracy of an object detector is called Intersection over Union (IoU), also known as the Jaccard index [48]. This index is used to compute the

similarity between finite sets. Suppose we have two sets A and B, which could be two AABBs. To measure similarity between them, IoU can be used and is calculated as follows,

$$IoU = J(A, B) = \frac{Area\ of\ Overlap}{Area\ of\ Union} = \frac{|A \bigcap B|}{|A \bigcup B|}. \tag{5.2}$$

Herein, we use IoU as the primary metric for detection accuracy. We also utilize mAP to provide a metric that can be compared to many results in the literature.

5.4.2.1 Augmentation Methods

In the experiments performed, the Python Imaging Library (PIL) is utilized to obtain augmented images. In PIL brightness and contrast enhancement classes, a factor is used for the change of brightness and contrast. When this factor is one, it gives the original image. Based on the experimental evaluation, in ADAS examples, factor values are chosen to be 1, 0.25, 0.5, 1.5, 2.0 and 2.5 for brightness and contrast. To add Gaussian noise, the noise variance is chosen to be 0.001, 0.003 and 0.005. This is based on qualitative image assessment, since these noise levels do not drastically alter the appearance of the input image. Other augmentation methods, including edge enhancement, global histogram equalization and Gaussian blurring (radius = 2), are predefined image operations in PIL.

5.4.2.2 Datasets

The primary research task is developing a robust system for ADAS, and two of the major objects that are crucial for detection in ADAS are traffic cones and pedestrians. In our specific industrial application, we also need to detect boxes, because they are commonplace items in our project's industrial setting. Therefore, we collect cone, pedestrian

and box images in controlled scenarios, which produce variations in different angles and distances. Specifically, the image dataset is collected using an FLIR Chameleon3 USB camera with a Fujinon 6-mm lens. Training images also include images taken by a Nikon D7000 camera and images from the PASCAL VOC dataset [22]. The total number of training images is around 22,000. For testing images, one cone, pedestrian or box is randomly placed or stands at different locations from 5 m to 20 m away and -20 degrees to +20 degrees in azimuth (left to right) from the camera, which covers the FLIR camera's full field of view horizontally and the detection range. For traffic cones, there are 287 testing images; for pedestrian, there are 208 testing images; and for boxes, there are 200 testing images. For generalization purposes, we also show results for the PASCAL VOC 2007 test dataset [22].

5.4.2.3 Training Parameters

The network structure for YOLO is basically the same as the default structure of YOLOv2 [86] except the last layer. YOLOv2 divides each image into 13 by 13 grids. For our application, we predict five base bounding boxes for each grid in the image. For each box, there are 4 numbers (for the top left and bottom right coordinates of the AABB), 1 confidence score and 5 class scores. Therefore, we change the filter size to $13 \times 13 \times 50$ ($5 \times (4 + 1 + 5) = 50$) in the last layer. We also change the learning rate to 10^{-5} to avoid divergence. We use a batch size of 64, a momentum of 0.9 and a decay of 0.0005, which are the same as those in the original YOLOv2 configuration.

For Faster R-CNN, the model is trained with a VGG net. Most parameters are set to be the same as the original Faster R-CNN [87]. The changes are as follows: the number of classes is modified to five; the number of outputs in the class score layer is modified to five; the number of outputs in the bounding box prediction layer is modified to 20 (4×5).

5.4.2.4 Results

Figure 5.10 is one example image for pedestrian detection with ground-truth AABB, three AABBs from different augmented images and fused AABB using the proposed method. In this image, the yellow AABB is the ground-truth, which is human-determined and hand-labeled. Three augmentation methods are chosen, which are global histogram equalization, changing contrast with factor two and changing brightness with factor 2.5. The AABBs obtained from these augmented images are shown in white. The fused AABB using the proposed AABBFI method is shown in pink.

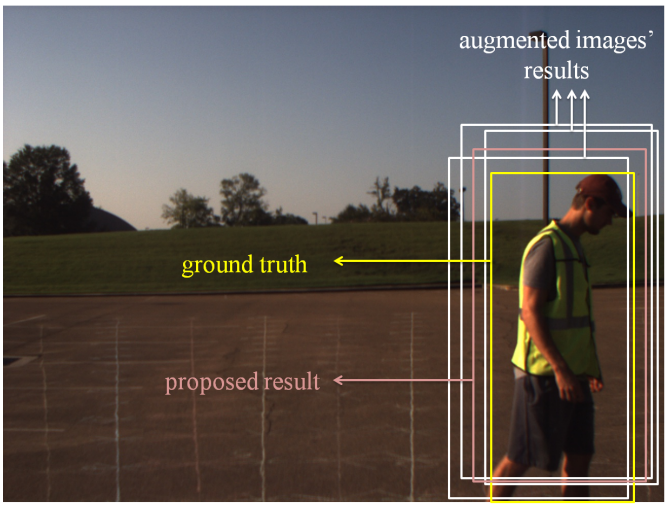


Figure 5.10

Fusion of three detection AABBs (white) into one AABB (pink), compared with the ground-truth (yellow).

The value of $g(\{x_2, x_3\})$ for the FM of agreement on the x-axis for cone detection is shown in Figure 5.11. From this figure, we can see that the value of the FM of agreement substantially changes from case to case. In other words, for each detection, the proposed fusion method gives different "weight" to each AABB, unlike the average operation, which gives the same "weight" around 0.33 when the number of AABBs to be fused is three.

In the experiment, we use a PC with an Intel i5-4460 dual-core CPU and one Nvidia GTX 1080 GPU. The computational overhead for AABBFI-based fusion is around 2.15 milliseconds (466 FPS), while for processing 10 augmented inputs, it is around 142.86 milliseconds (7 FPS). The fusion overhead is small compared to the overhead for using multiple inputs. The computational load is linear with the number of given inputs. More discussion about computational complexity is given in Section 5.4.3.

All ADAS experiment results are shown in Table 5.1. In this table, the "No fusion" column means that the results are detection outputs from the original images without any fusion method applied. Under the "With fusion" section, four fusion methods are listed: NMS, average, median and the proposed AABBFI. For all five fusion/non-fusion methods, the average IoU obtained and the number of detections are listed. We find that for cones, pedestrians and boxes, the highest accuracy and detection happen when using the proposed AABBFI method together with YOLO. For the PASCAL VOC dataset [22], we use the threshold of 0.25 for the confidence score to calculate mAP, and we find that after fusion, not only the average IoU increases, but also mAP increases.

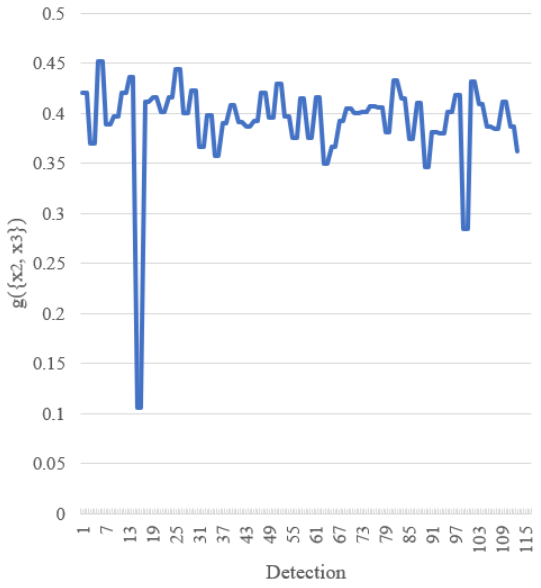


Figure 5.11

Value changes of $g(\{x_2, x_3\})$ for the FM of agreement on the x-axis for cone detection.

In Table 5.1, we also include results on detecting pedestrians using Faster R-CNN [87]. Comparing the results, it is shown that YOLO outperforms Faster R-CNN both in accuracy and detection. Moreover, in the case that we choose Faster R-CNN as the detector, the AABBFI method does not help with accuracy compared with the average and median methods. Furthermore, Table 5.1 shows that mAP improves by using fusion. The mAP improved the most by using the proposed method. In table 5.1, the best results are shown in bold.

Table 5.1

ADAS experimental results. Baseline means there is no fusion performed, that is, only the original image is processed.

	No fusion	**With fusion**			
Fusion method	**Baseline**	**NMS**	**Average**	**Median**	**Proposed**
YOLO Cone					
Average IoU	0.6199	0.6676	0.6676	0.6717	**0.6763**
Detection	260/287	**287/287**	**287/287**	**287/287**	**287/287**
YOLO Box					
Average IoU	0.6722	0.6756	0.6925	0.6872	**0.7031**
Detection	**200/200**	**200/200**	**200/200**	**200/200**	**200/200**
YOLO Pedestrian					
Average IoU	0.7727	0.7023	0.7995	0.7966	**0.8141**
Detection	**208/208**	**208/208**	**208/208**	**208/208**	**208/208**
Faster R–CNN Pedestrian					
Average IoU	0.7402	0.7377	0.7660	0.7581	0.7499
Detection	206/208	206/208	206/208	206/208	206/208
YOLO VOC 2007 All					
Average IoU	0.6106	0.6654	0.6678	0.6728	**0.6746**
mAP	71.10%	72.36%	72.82%	72.41%	**72.87%**

5.4.3 Discussion

From the synthetic results, it is clear that with the presence of possible or obvious outliers, the proposed AABBFI method can fuse the AABBs in a way more similar to our expectation, and it also can reduce or eliminate the effects of possible or obvious outliers.

From the ADAS experimental results, there is improvement both on IoU and detection by using the proposed AABBFI method together with YOLO. Specifically, in most cases, fusion methods help with increasing detection. It is also shown that comparing with input images' average IoU without fusion, by using the average, median and proposed fusion methods, the average IoU results increase. Comparing NMS, average, median and the proposed fusion operation, the proposed AABBFI method gives the average IoU the most improvement.

From Figure 5.11, we can see that unlike the static operations such as average and median, the proposed fusion method is dynamic. The FM of agreement substantially changes from case to case. Consequently, the fusion is done in a detection-by-detection and object-by-object manner. For fusion, we evaluate the FM of agreement among AABBs for each detection and fuse them using a computationally-intelligent method named AABBFI. As mentioned in Section 5.1, the fusion for detection in this scenario cannot be learned in general and is done in a case-by-case manner in this paper.

For the proposed AABBFI method, it works when the networks have variety in the results. Faster R-CNN is a region proposal-based method, which means that it uses part of its network to propose possible regions that could have objects we want to detect, and this may reduce the variety in the output. On the other hand, YOLO predicts both AABBs

and labels through one giant network, which makes it fast. Furthermore, this kind of one–shot network design may produce more variations, which leads to more accurate output by using the AABBFI method. As a result, we choose YOLO together with the AABBFI method in the proposed system. The results from the ADAS experiments show that in cone, pedestrian and box detection, the proposed method gives the best results.

In the ADAS examples, the improvement of both detection and IoU could come from two aspects. First, variation can lead to detection. In the original image, the detector might not be able to detect cones, pedestrians or boxes, but in its variations, the detector may work. Second, fusion based on agreement stabilizes the finally AABB. AABBFI is a dynamic fusion method that produces different result when the "agreement" among inputs changes. NMS is also a dynamic fusion method, but the results using it are not as good as the proposed fusion method. On the other hand, the average and median operations are both static, which means that they do not consider the specific characteristics of each input.

To guarantee a real-time ADAS, we expect that the fusion adds as little computational complexity as possible to the original detection system. The computational complexity of the proposed AABBFI (fusion only) is actually tiny, which is $O(n)$, where n stands for the number of input images. It is the same as NMS, average and median operation. This means that the four fusion methods all have the same low computational complexity and do not add much computational burden to the system. Another thing that may slow down the system is there are multiple augmented inputs that need to be processed. We can avoid the slowdown either via multiple processing units or a more powerful processing unit. This highlight a trade-off in many systems: there are multiple approaches, some more

computationally complexity, and some less computationally complexity. Given sufficient hardware resources to keep an appropriate frame rate, the proposed solution is feasible. The computational overhead in Section 5.4.2.4 shows that with ten augmented images, using our hardware configuration, we can achieve real-time performance in our definition (5 Hz or above).

In summary, from ADAS examples, we find that AABB-based fusion with YOLO produces the highest accuracy and detection results compared with other methods listed. From the synthetic example, we could obtain some rationality behind this improvement. The proposed system increases the accuracy in the detection stage while maintaining the real-time characteristic in the original detection system, which makes this a realistic solution for a real-time domain like ADAS.

5.5 Conclusions

This Chapter proposed a computational intelligence system for more accurate object detection in real time. This system uses augmentation methods before a deep learning detector and then an AABB fuzzy integral (AABBFI) on the resultant AABBs. Three synthetic examples show the rationality of using the AABBFI rather than NMS/average/median operation. Experimental ADAS examples show that for real-world datasets, the proposed system gives the highest accuracy and detection results, without adding much computational complexity for real-time purposes. Our proposed system is not only fast, but also accurate, which are two important criteria in ADAS. By using this computational intelligence system, we are able to build a more robust object detection sub-system for ADAS

applications, with the proposed system showing improvement in both IoU and mAP metrics. Furthermore, very good results were obtained when only utilizing three combined inputs, making the computational load roughly three-times the load for only using one input (the original image).

CHAPTER 6

LIDAR AND CAMERA DETECTION FUSION IN A REAL-TIME INDUSTRIAL MULTI-SENSOR COLLISION AVOIDANCE SYSTEM

6.1 Introduction

Collision avoidance systems are important for protecting people's lives and preventing property damage. Herein, we present a real–time industrial collision avoidance sensor system, which is designed to not run into obstacles or people and to protect high–valued equipment. The system utilizes a scanning LiDAR and a single RGB camera. A passive beacon is utilized to mark off quarantined areas where the industrial vehicle is not allowed to enter – thus preventing collisions with high–valued equipment. A front–guard processing mode prevents collisions with objects directly in front of the vehicle.

To provide a robust system, we utilize a Quanergy eight-beam LiDAR and a single RGB camera. The LiDAR is an active sensor, which can work regardless of the natural illumination. It can accurately localize objects via their 3D reflections. However, the LiDAR is monochromatic and can't differentiate objects based on color. Furthermore, for objects that are far away, the LiDAR may only have one to two beams intersecting the object, making reliable detection problematic. Unlike the LiDAR, RGB cameras can make detection decisions based on texture, shape and color. An RGB stereo camera can be used to detect objects and to estimate 3D positions. However, stereo cameras require extensive

processing and often have difficulty estimating depth when objects lack textural cues. On the other hand, a single RGB camera can be used to accurately localize objects in the image itself (e.g. determine bounding boxes and classify objects). However, the resulting localization projected into 3D space is poor compared to the LiDAR. Furthermore, the camera will degrade in foggy or rainy environments, whereas the LiDAR can still operate effectively. Herein, we utilize both the LiDAR and the RGB camera to accurately detect (e.g. identify) and localize objects. The focus of this paper is the fusion method, and we also highlight the LiDAR detection since it is not as straightforward as the camera–based detection. The contributions of this paper are:

- We propose a fast and efficient method that learns the projection from the camera space to the LiDAR space and provides camera outputs in the form of LiDAR detection (distance and angle).

- We propose a multi–sensor detection system that fuses both the camera and LiDAR detections to obtain a more accurate and robust beacon detection.

- The proposed solution has been implemented using a single Jetson TX2 board (dual CPUs and a GPU) board to run the sensor processing and a second TX2 for the Model Predictive Control (MPC) system. The sensor processing runs in real–time (5 Hz).

- The proposed fusion system has been built, integrated, and tested using static and dynamic scenarios in a relevant environment. Experimental results are presented to show the fusion efficacy.

This paper is organized as follows: Section 6.2 introduces LiDAR detection, camera detection, and fusion of LiDAR and camera. Section 6.3 discusses the proposed fusion system, and Section 6.4 gives an experimental example showing how fusion and the system

work. In Section 6.5, we compare the results with and without fusion. Finally, Section 6.6 contains conclusions and future work.

6.2 Background

6.2.1 Camera detection

Object detection from camera imagery involves both classification and localization of each object that we have interest in. We do not know ahead of time how many objects we expect to find in each image, which means that there is a varying number of outputs for every input image. We also do not know where these objects may appear in the image, or what their sizes might be. As a result, the object detection problem becomes quite challenging.

With the rise of deep learning (DL), object detection methods using DL [85, 86, 87, 97, 13, 61] have surpassed many traditional methods [14, 25] in both accuracy and speed. Broadly speaking, there are two approaches for image object detection using DL. One approach is based on region proposals. Faster R-CNN [87] is one example. This method first runs the entire input image through some convolutional layers to obtain a feature map. Then there is a separate region proposal network, which use these convolutional features to propose possible regions for detection. Last, the rest of the network gives classification to these proposed regions. As there are two parts in the network, one for predicting the bounding box, and the other for classification, this kind of architecture may significantly slow down the processing speed. Another type of approach uses one network for both predicting potential regions and for label classification. One example is YOLO [85, 86]. Given an input image, YOLO first divides the image into coarse grids. For each grid, there

is a set of base bounding boxes. For each base bounding box, YOLO predicts offsets off the true location, a confidence score, and classification scores if it thinks that there is an object in that grid location. YOLO is fast, but sometimes can fail to detect small objects in the image.

6.2.2 LiDAR detection

For LiDAR detection, the difficult part is classifying points based only on a sparse 3D point cloud. One approach [58] uses eigen–feature analysis of weighted covariance matrices with a Support Vector Machine (SVM) classifier. However, this method is targeted at dense airborne LiDAR point clouds. In another method [31], the feature vectors are classified for each candidate object with respect to a training set of manually labeled object locations. With its rising popularity, DL has also been used on 3D object classification. Most DL–based 3D object classification problems involve two steps: deciding a data representation to be used for the 3D object and training a convolutional neural network (CNN) on that representation of the object. VoxNet [71] is a 3D CNN architecture for efficient and accurate object detection from LiDAR and RGBD point clouds. An example of DL for volumetric shapes is the Princeton ModelNet dataset [105], which has proposed a volumetric representation of the 3D model and a 3D volumetric CNN for classification. However, these solutions also depend on a high density (high beam count) LiDAR, so they would not be suitable for system with an eight-beam Quanergy M8.

6.2.3 Camera and LiDAR detection fusion

Different sensors for object detection have their advantages and disadvantages. Sensor fusion integrates different sensors for more accurate and robust detection. For instance, in object detection, cameras can provide rich texture–based and color–based information, which LiDAR generally lacks. On the other hand, LiDAR can work in low visibility, such as at night or in moderate fog or rain. Camera processing in poor–weather conditions can degrade or even fail completely. Also, for the detection of the object position relative to the sensor, the LiDAR can provide a much more accurate spatial coordinate estimation compared to a camera. As both camera and LiDAR have their advantages and disadvantages, when fusing them together, the ideal algorithm should fully utilize their advantages and eliminate their disadvantages.

One approach for camera and LiDAR fusion uses extrinsic calibration. Some works of this approach [33, 81, 26] use various checkerboard patterns or look for corresponding points or edges in both the LiDAR and camera imagery. Other works of this approach [54, 32, 78, 80, 7] look for corresponding points or edges in both the LiDAR and camera imagery in order to perform extrinsic calibration. However, for the majority of works of this approach, the LiDARs they use have either 32 or 64 beams, which provide relatively high vertical spatial resolution, but are prohibitively expensive. Another example is [56], which estimates transformation matrices between the LiDAR and the camera. Other similar examples include [109, 99]. But these works are only suitable for modeling of indoor and short range environments. Another approach uses a similarity measure. One example is [70], which automatically registers LiDAR and optical images. This approach also uses

dense LiDAR measurements. A third kind of approach uses stereo cameras and LiDAR for fusion. One example is [65], which fuses sparse 3D LiDAR and dense stereo image point clouds. However, the matching of corresponding points in stereo images is computationally complex and may not work well if there is little texture in the images. Both of these approaches require dense point clouds and would not be effective with the smaller LiDARs such as the Quanergy M8. Comparing with previous approaches, the proposed approach in this paper is unique in two aspects: (1) a single camera and relatively inexpensive eight–beam LiDAR are used for an outdoor collision avoidance system, and (2) the proposed system is a real–time system.

6.2.4 Fuzzy logic

When Zadeh invented "fuzzy" to describe the ambiguities in the world, he developed a language to describe his ideas. One of the important terms in "fuzzy" is fuzzy set, which is defined as a class of objects with a continuum of grades of membership [106]. The relation between linguistic variable and fuzzy set is described as "a linguistic variable can be interpreted using fuzzy sets" in [89]. This statement basically means a fuzzy set is the mathematical representation of linguistic variables. An example of fuzzy set is "a class of tall students". It is often denoted as $\underset{\sim}{A}$ or sometimes A. The membership function assigns to each object a grade of membership ranging between zero and one [106].

Fuzzy logic is also proposed in [106] by Zadeh. Instead of giving either true or false conclusions, fuzzy logic uses degrees of truth. The process of fuzzy logic includes three steps. First, input is fuzzified into fuzzy membership function. Then, IF-THEN rules are

applied to produce the fuzzy output function. Last, the fuzzy output function is de-fuzzified to get specific output values. Thus, fuzzy logic can be used to reason with inputs that have uncertainty.

Zhao et al. [110] applied fuzzy logic to fuse Velodyne 64–beam LiDAR and camera data. They utilized LiDAR processing to classify objects and then fused the classifications based on the bounding box size (tiny, small, middle or large), the image classification result, the spatial and temporal context of the bounding box, and the LiDAR height. Their system classified outputs as greenery, middle or obstacle. Again, the use of a Velodyne 64–beam LiDAR is prohibitively expensive for our application.

Fuzzy logic provides a mathematically well–founded mechanism for fusion with semantic meaning, such as "the LiDAR detects an object with *high confidence*". This semantic statement can be quantified mathematically by the fuzzy function. The fuzzy logic system can then be used to fuse the fuzzified inputs, using the rules of fuzzy logic. Herein, we apply fuzzy logic to combine confidence scores from the camera and the LiDAR to obtain a detection score for the final fusion result.

6.3 Proposed system

6.3.1 Overview

The proposed system architecture (block diagram) is shown in Figure 6.1. In the figure, the Camera and LiDAR are exteroceptive sensors, and their outputs go to the Signal Processing boxes. The LiDAR signal processing is discussed in section 6.3.6 and the camera signal processing is discussed in section 6.3.4. The LiDAR detects beacons and rejects other objects. The LiDAR signal processing output gives beacon locations as a distance

in meters (in front of the industrial vehicle), the azimuth angle in degrees, and a discriminant value, which is used to discriminate beacons from non–beacons. The camera reports detections as relative bounding box locations in the image, as well as a confidence of the detection and classification. The LiDAR and camera information is fused to create more robust detections. The fusion is discussed in section 6.3.7.1. Figure 6.2 shows one application of using this system for collision avoidance. In this figure, the green area in the middle is quarantined and designated as a non–entry area for industrial vehicles.

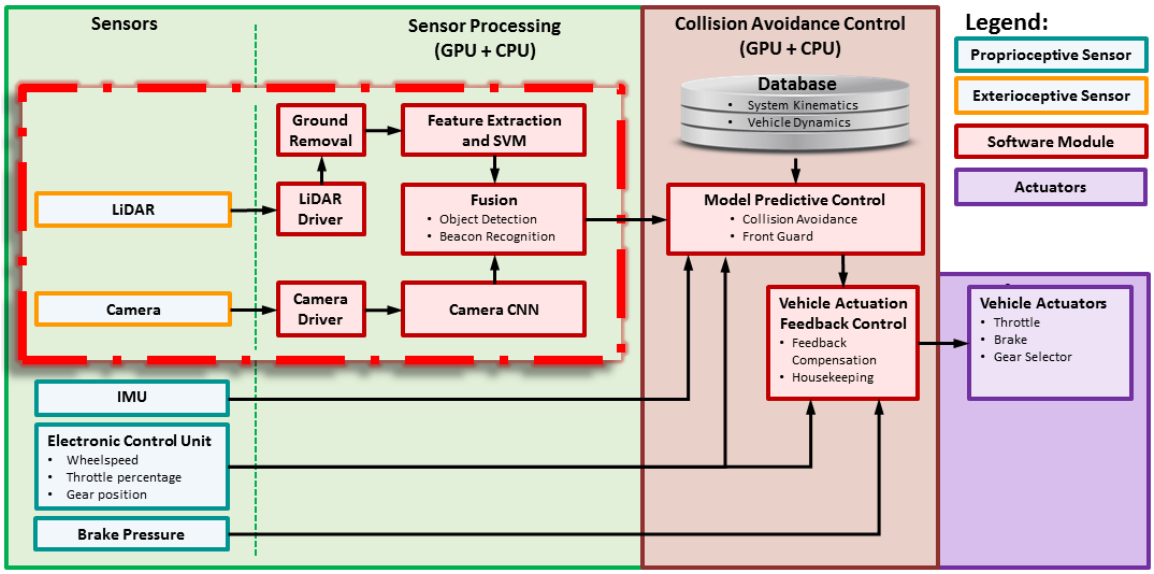


Figure 6.1

Collision avoidance system block diagram. IMU = Inertial Measurement Unit. GPU = Graphics Processing Unit. CPU = Central Processing Unit. SVM = Support Vector Machine. CNN = Convolutional Neural Network. Figure best viewed in color.

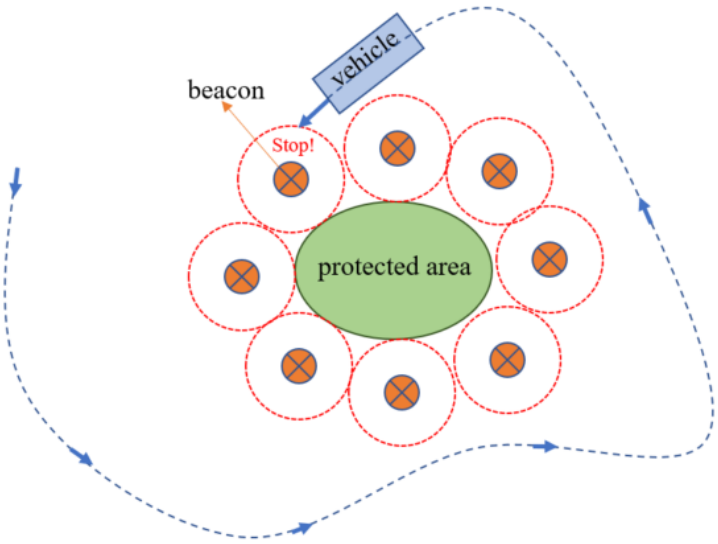


Figure 6.2

Using passive beacons to delineate a no–entry area.

As shown in Figure 6.1, the vehicle also has proprioceptive sensors, including wheel speed, steering angle, throttle position, engine control unit (ECU), two brake pressure sensors, and an inertial measurement unit (IMU). These sensors allow the vehicle actuation and feedback control system to interpret the output of the system model predictive control (MPC) system and send commands to the ECU to control throttle and breaking.

The MPC and vehicle actuation systems are critical to controlling the industrial vehicle, but are not the focus of this paper. These systems model the vehicle dynamics and system kinematics, and internally maintain a virtual barrier around the detected objects. MPC by definition is a model–based approach which incorporates a predictive control methodology to approximate optimal system control. Control solutions are determined through the evaluation of a utility function comparing computations of a branching network of future

outcomes based on observations of the present environment [60, 2, 46]. The implemented MPC algorithm, to which this work is applied, predicts collision paths for the vehicle system with detected objects, such as passive beacons. Vehicle speed control solutions are computed for successive future time steps up to the prediction horizon. A utility function determines optimal control solutions from an array of possible future control solutions. Optimal control solutions are defined in this application as solutions which allow for operation of the vehicle at a maximum possible speed defined as the speed limit which ensures the vehicle is operating within the control limitations of brake actuation subsystem such that a breach of a virtual boundary around detected objects may be prevented. In the implemented system, the prediction horizon may exceed 5 seconds. At the maximum possible vehicle speed this horizon is consistent with detection horizon of the sensor network. The development of the system model is partially described in [62] and [15].

The proposed fusion system is shown in Figure 6.3. In this system, we first obtain images from the camera, then the through the camera object detection system, we estimate the bounding box coordinates of beacons together with the confidence scores for each bounding box. Next, the detection bounding box from the single camera is mapped through a neural network (NN) into a distance and angle estimate, in order to be able to fuse that information with the distance and angle estimates of the LiDAR processing.

On the other side, we have point cloud information from the LiDAR, then through LiDAR data processing, we estimate the distance and angle information of the detected beacons by the LiDAR. Through the LiDAR processing, we also obtain a pseudo-confidence score. With the LiDAR detection results in the form of distance and angle from both cam-

era and LiDAR, we fuse them together to obtain a final distance, angle and confidence score for each detection. We give more details about this in Section 6.3.8.

In our application, the maximum speed of the vehicle is about 5 meters per second. In order to have enough reaction time for collision avoidance, the detection frequency needs to be 5 Hz or above. We accomplish this requirement for sensor processing on one NVIDIA Jetson TX2, and use another TX2 for the MPC controller. More details are given in Section 6.3.2.

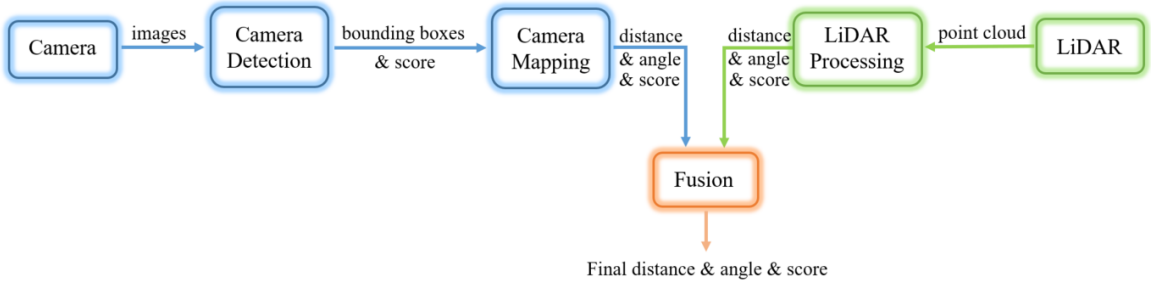


Figure 6.3

Proposed fusion system high–level block diagram.

6.3.2 Real-time implementation

To achieve the desired 5 Hz update rate for real-time performance, multiple compromises were made to improve software speed enough to meet the requirements. Notably, a naïve approach for removal of ground points from the LiDAR point cloud was used instead of a more complex, and accurate, method. Furthermore, a linear SVM, with a limited feature set, was used as opposed to a kernel SVM or other non-linear methods. The specifics of these methods are covered in section 6.3.6.

To manage intra–process communication between sensor drivers and signal processing functions, the Robot Operating System (ROS) was used [82]. ROS provides already–implemented sensor drivers for extracting point cloud and image data from the LiDAR and camera, respectively. Additionally, ROS abstracts and simplifies intra-process data transfer by using virtualized Transmission Control Protocol (TCP) connections between the individual ROS processes, called nodes. As a consequence of this method, multiple excess memory copies must be made, which begin to degrade performance with large objects like LiDAR point clouds. To ameliorate this effect, ROS Nodelets were used for appropriate LiDAR processes. Nodelets builds on ROS Nodes to provide a seamless interface for pooling multiple ROS processes into a single, multi–threaded process. This enables efficient zero–copy data transfers between signal processing steps for the high bandwidth LiDAR computations. This proved essential due to the limited resources of the TX2, with Central Processing Unit (CPU) utilization averaging at over 95% even after Nodelet optimization.

In Figure 6.4, the high–level flowchart of the relevant ROS nodes can be seen along with their execution location on the Jetson TX2's hardware. The NVIDIA Jetson TX2 has a 256 CUDA (Compute Unified Device Architecture) core GPU, a Dual CPU, 8 GB of memory and supports various interfaces including Ethernet. As the most computationally intensive operation, the camera CNN has the entire Graphical Processing Unit (GPU) allocated to it. The sensor drivers, signal processing, and fusion nodes are then relegated to the CPU. Due to the high core utilization from the LiDAR, care has to be taken that enough CPU resources are allocated to feed images to the GPU or significant throttling of the CNN can occur.

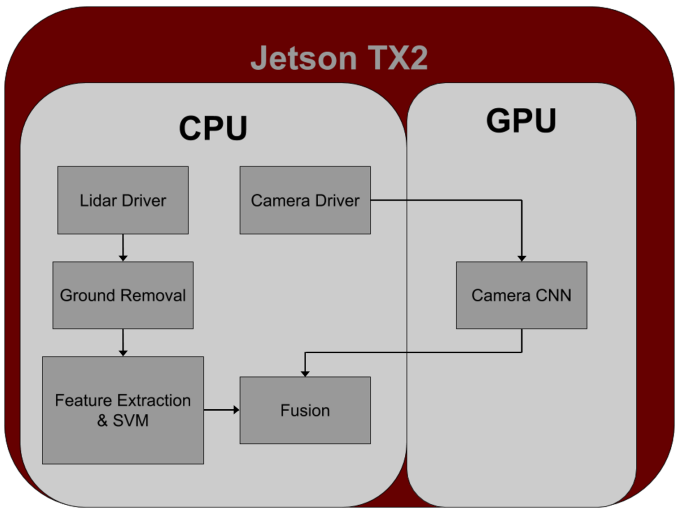


Figure 6.4

ROS Dataflow on Jetson TX2.

6.3.3 Beacon

The beacon is constructed of a high–visibility 28” orange traffic cone with a 2” diameter highly–reflective pole extending two meters vertically. A beacon is shown in Figure 6.5. The beacon presents as a series of high intensity values in a LiDAR scan, and thus provides a high signal level compared to most background objects. A beacon delineates an area in the industrial complex that is off limits to the vehicle—usually to protect high–valued assets.

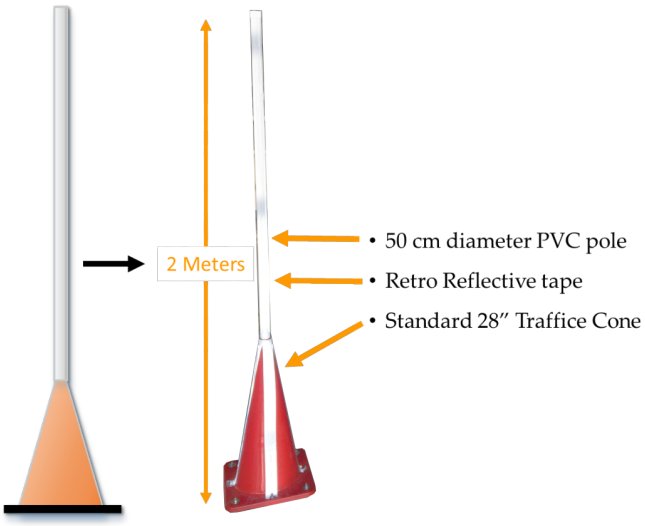


Figure 6.5

Beacon.

6.3.4 Camera detection.

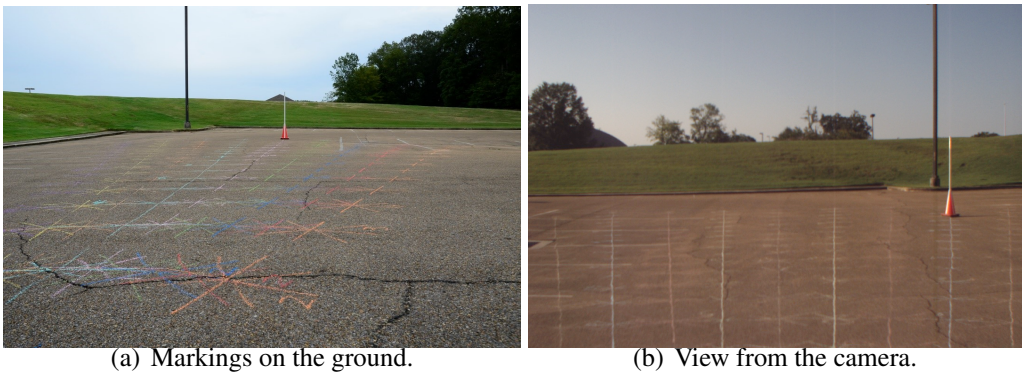
Herein, we apply a deep-learning model called YOLO [85, 86] to detect objects present in the single camera image. YOLO outperforms most other deep learning methods in speed, which is crucial for our application. YOLO achieves such speed by combining both region and class prediction into one network, unlike region–based methods [13, 87, 30], which have separate network for region proposals.

A large training dataset of beacon images presented in Section 6.4 was collected on different days, different times of day and weather conditions (e.g. full sun, overcast, etc.), and was hand–labeled by our team members. Figure 6.6 shows how we draw bounding boxes and give labels on an image from camera.

Figure 6.6

Labeling for camera detection.

The dataset we collected covers the camera's full field of view (FOV) ($-20°$ to $20°$ azimuth), and detection range (5 to 40 meters) of the camera (FLIR Chameleon3 USB camera with Fujinon 6mm lens). It reliably represents the detection accuracy from different angles and distances. Figure 6.7 shows the markings on the ground that help us to locate the beacon at different angles and distances. These tests were performed early in the project to allow us to estimate the LiDAR detection accuracy.



(a) Markings on the ground. (b) View from the camera.

Figure 6.7

Markings on the ground covering the full FOV and detection range of the camera.

All of the beacons have a highly consistent construction, with near–identical size, shape, and coloring. These characteristics allow the detection algorithm to reliably recognize beacons at a range of up to 40 meters under a variety of lighting conditions. Figure 6.8 shows one example of camera detection of beacons.

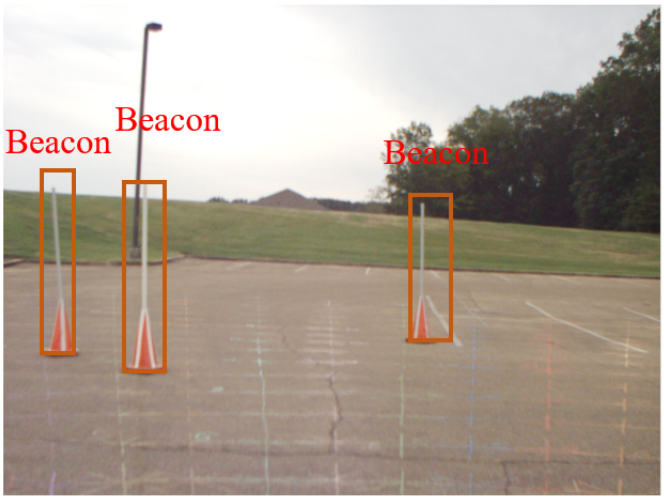


Figure 6.8

Camera detection example.

6.3.5 LiDAR front guard

The LiDAR detection mode is designed to detect beacons. However, other objects can be in the direct or near–direct path of the industrial vehicle. A front guard system is implemented with the LiDAR. After ground point removal, any remaining points within a rectangular solid region directly in front of the vehicle will be detected and reported to the control system.

6.3.6 LiDAR beacon detection

This section discusses the LiDAR's beacon detection system.

6.3.6.1 Overview

The LiDAR data we collect are in the form of a three-dimensional (3D) and 360 degree field–of–view (FOV) point cloud, which consists of x, y, z coordinates, along with

intensity and beam number (sometimes called ring number). The coordinates x, y, and z represent the position of each point relative to the origin centered within the LiDAR. Intensity represents the strength of returns and is an integer for this model of LiDAR. Highly reflective objects such as metal will have higher intensity values. The beam number represents in which beam the returned point is located. The LiDAR we use sends out eight beams. Figure 6.9 shows one scan of LiDAR point cloud. The beam numbers are shown in Figure 6.10. Assuming the LiDAR is mounted level to the ground, then beam 7 is the top beam, and is aimed upward approximately $3°$. Beam 6 points horizontally. The beams are spaced approximately $3°$ apart. Figure 6.9 shows one scan of LiDAR point cloud. In this figure, the ground points are shown with black points. The beacon presents as a series of closely–spaced points in the horizontal direction, as shown in the square.

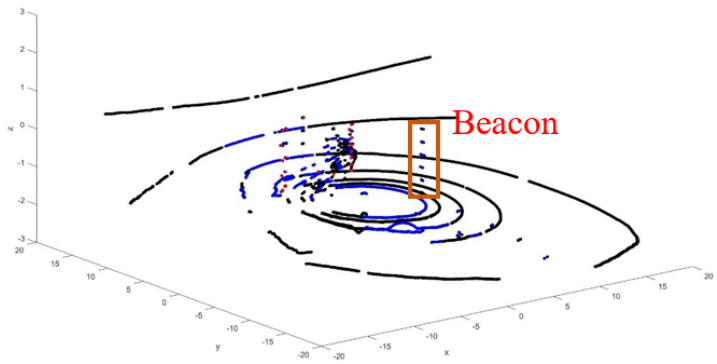


Figure 6.9

Detection via LiDAR.

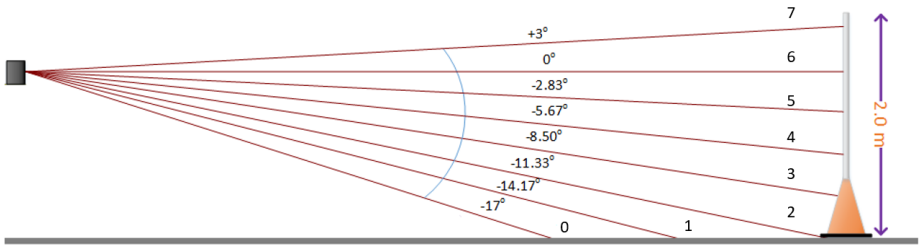


Figure 6.10

LiDAR beam spacing visualization and beam numbering. The LiDAR is the small black box on the left. Beam 6 is the horizontal beam (assuming the LiDAR is level).

General object detection in LiDAR is a difficult and unsolved problem in the computer vision community. Objects appear at different scales based on their distance to the LiDAR, and they can be viewed from any angle. However, in our case, we are really only interested in detecting the beacons, which all have the same geometries. In our application, the industrial vehicle can only go so fast, so that we only need to detect beacons about 20 meters away or closer. The beacons are designed to have very bright returns. However, other objects can also have bright returns in the scene, such as people wearing safety vests with retro-reflective tape, or other industrial vehicles. Herein, we propose a system that first identifies clusters of points in the scene that present with bright returns. Next, a linear SVM classifier is used to differentiate the beacon clusters from non-beacon clusters.

The LiDAR detection system works by utilizing intensity–based and density–based clustering (a modified DBSCAN algorithm) on the LiDAR point cloud. The system then examines all points near the cluster centroid by extracting features and using a linear SVM

to discriminate beacons from non-beacons. Details of these algorithm are discussed in the subsections below.

6.3.6.2 LiDAR clustering

A modified DBSCAN clustering algorithm [21], which clusters based on point cloud density as well as intensity is used to cluster the bright points, as shown in algorithm 4. The cluster parameter ϵ was empirically determined to be 0.5m based on the size of the beacon. Values much larger could group two nearby beacons (or a beacon and another nearby object) together into one cluster, and we wanted to keep them as separate clusters.

In Equation 4, the distances are estimated using Euclidean distances with only the x (front-to-back) and y (left-to-right) coordinates. Effectively, this algorithm clusters bright LiDAR points by projecting them down onto the $x - y$ plane. Alternately, all three coordinates, x, y, z could be used, but the features don't require this because they will be able to separate tall and short objects. This approach is also more computationally efficient than using all three coordinates in the clustering algorithm.

Algorithm 4 LiDAR bright pixel clustering.

input: LiDAR high–intensity point cloud $P_{HT} = \{x_j, y_j, z_j, i_j, r_j\}$.

input: LiDAR low–intensity point cloud $P_{LT} = \{x_j, y_j, z_j, i_j, r_j\}$.

input: Inner region x-extent: Δx_I (meters).

input: Inner region y-extent: Δy_I (meters).

input: Outer region x-extent: Δx_O (meters).

input: Outer region y-extent: Δy_O (meters).

input: LiDAR height above ground: $Z_L = 1.4$ (meters).

output :Feature vector **f**.

Cluster the high–intensity point cloud:

for e **do**ach point in **p** in the high–intensity point cloud

Cluster points and determine cluster centers.

end for

Calculate features:

for each cluster center point $c = (x_C, y_C, z_C)$ in the point cloud **do**

Determine all points in P_{HT} in the inner region using Equation 6.1 and calculate feature 1 from Table 6.1.

Determine all points in P_{HT} in the outer region using Equation 6.2 and calculate feature 4 from Table 6.1.

Determine all points in P_{LT} in the inner region using Equation 6.1 and calculate features 6,7,9 and 10 from Table 6.1.

Determine all points in P_{LT} in the outer region using Equation 6.2 and calculate features 2,3,5 and 8 from Table 6.1.

end for

Return **f** $= [f_1, f_2, f_3, \cdots, f_{20}]$.

6.3.6.3 LiDAR feature extraction

To extract the features, the ground points must be removed before feature processing occurs, or else false alarms can occur. Since this industrial application has a smooth, flat area for the ground, we employ a simple vertical threshold to remove ground points. If

this system were to be generalized to areas with more varying ground conditions, ground estimation methods such as those in [101, 73, 91, 84, 10, 74] could be utilized. However, for this particular application, this was not necessary.

Next, the point intensities are compared to an empirically determined threshold. The beacon is designed so that it provides bright returns to the LiDAR via the retro-reflective vertical pole. This works well, but there are also other objects in the scene that can have high returns, such as other industrial vehicles with retro–reflective markings or workers wearing safety vests with retro–reflective stripes. In order to classify objects as beacons and non–beacons, hand–crafted features are utilized (these are discussed below). After ground–removal and thresholding the intensity points, we are left with a set of bright points. A second set of intensity points is also analyzed, which consists of all of the non–ground points (e.g. the point cloud after ground removal). The points in the bright point cloud are clustered. Beacons appear as tall, thin objects, whereas all other objects are either not as tall, or wider. We extract features around the cluster center in a small rectangular volume centered at each object's centroid. We also extract features using a larger rectangular volume also centered around the objects centroid. Features include counting the number of bright points in each region, determining the x, y, and z extents of the points in each region, etc. Beacons mainly have larger values in the smaller region, while other objects have values in the larger regions.

The two regions are shown in Figure 6.11. The idea of using an inner and an outer analysis region is that a beacon will mostly have bright points located in the inner analysis region, while other objects, such as humans, other industrial vehicles, etc., will extend into

the outer regions. Equations 6.1 and 6.2 define whether a LiDAR point p_j with coordinates (x_j, y_j, z_j) is in the inner region or outer region, respectively, where the object's centroid has coordinates (x_C, y_C, z_C). Reference Figure 6.11 for a top–down illustration of the inner and outer regions.

Figure 6.11 shows an example beacon return with the analysis windows superimposed. Both the inner and outer analysis regions have x and y coordinates centered at the centroid location. The inner analysis region has depth (x coordinate) of 0.5 meters, width (y coordinate) of 0.5 meters, and the height includes all points with z coordinate values of -1.18 meters and above. The outer region extends 2.0 meters in both x and y directions and has the same height restriction as the inner region. These values were determined based on the dimensions of the beacon and based on the LiDAR height. The parameters Δx_I, Δy_I and z_{MIN} define the inner region relative to the centroid coordinates. Similarly, the parameters Δx_O, Δy_O and z_{MIN} define the outer region relative to the centroid coordinates. A point is in the inner region if

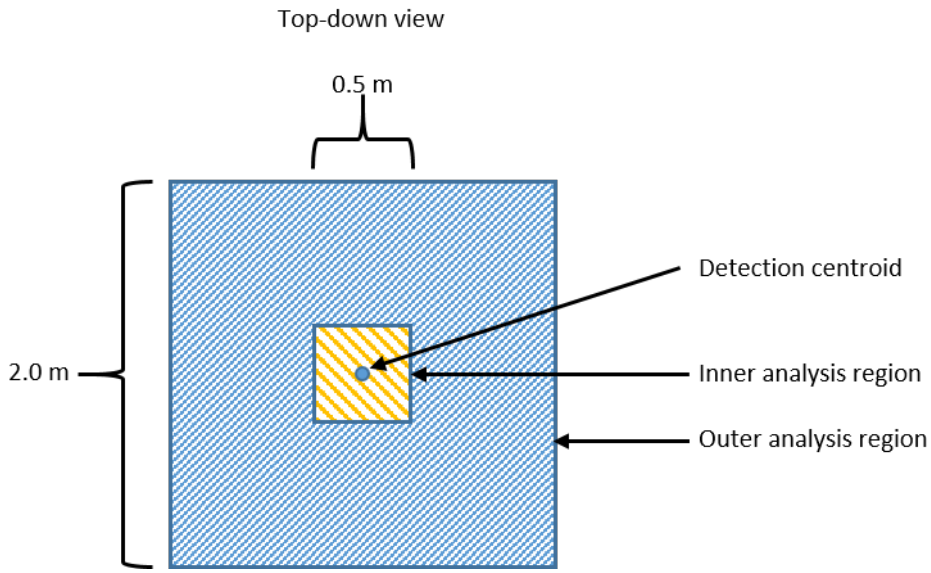


Figure 6.11

LiDAR detection regions (inner and outer) visualized from a top–down view.

$$\left(x_C - \frac{\Delta x_I}{2}\right) \leq x_j \leq \left(x_C + \frac{\Delta x_I}{2}\right) and$$
$$\left(y_C - \frac{\Delta y_I}{2}\right) \leq y_j \leq \left(y_C + \frac{\Delta y_I}{2}\right) and \tag{6.1}$$
$$z_{MIN} \geq z_j,$$

and a point is in the outer region if

$$\left(x_C - \frac{\Delta x_O}{2}\right) \leq x_j \leq \left(x_C + \frac{\Delta x_O}{2}\right) and$$
$$\left(y_C - \frac{\Delta y_O}{2}\right) \leq y_j \leq \left(y_C + \frac{\Delta y_O}{2}\right) and \tag{6.2}$$
$$z_{MIN} \geq z_j.$$

In order to be robust, we also extract features on the intensity returns. A linear SVM was trained on a large number of beacon and non–beacon objects, and each feature was sorted based on its ability to distinguish beacons from non–beacons. In general, there is a marked increase in performance then the curve levels out. It was found that ten features were required to achieve very high detection rates and low false alarms. These ten features were utilized in real time, and the system operates in real time at the frame rate of the LiDAR, 5 Hz. The system was validated by first running on another large test set independent of the training set, with excellent performance as a result as presented in Section 6.3.6.4. Then, extensive field tests were used to further validate the results as presented in Section 6.4.

After detection, objects are then classified as beacons or non–beacons. These are appended to two separate lists and reported by their distance in meters from the front of the industrial vehicle, their azimuth angle in degrees, and their discriminate value. Through extensive experimentation, we can reliably see beacons in the LiDAR's FOV from 3 to 20 meters.

LiDAR feature extraction follows the overall algorithm shown in Algorithm 5. The input is the LiDAR point cloud $P = \{x_j, y_j, z_j, i_j, r_j\}$, where j is the index variable for the j–th point, x, y, z, i and r refer to the x point in meters, the y point in meters, the z point in meters, the intensity and the beam number, respectively, for the j–th point. Note all coordinates are relative to the center of the LiDAR at point $(0, 0, 0)$. The x coordinate is positive in front of the LiDAR, and negative behind. The y coordinate is positive to the

left of the LiDAR and negative to the right. The z coordinate is positive above the LiDAR and negative below.

In order to extract features for objects, any points that did not provide a return to the LiDAR are removed. These points will present as NaN's (Not a Number) in the LiDAR point cloud. Next, the estimated ground points are removed. The non–ground points are divided into two data subsets: The high–threshold (HT) data and the low–threshold (LT) data. The HT data points only contain high–intensity returns, while the LT data contains points greater than or equal to the low–intensity threshold. Both data subsets do not contain any ground points. The high–intensity threshold was set to 15, and the low–intensity threshold to 0. These threshold values were determined experimentally based on examination of multiple beacon returns at various distances.

Algorithm 5 LiDAR high–level feature extraction preprocessing.

input: LiDAR point cloud $P = \{x_j, y_j, z_j, i_j, r_j\}$.

input: Low–intensity threshold: T_L.

input: High–intensity threshold: T_H.

input: Ground Z threshold: T_G (meters).

output: Feature vector **f**.

Remove non–return points:

for each point in the point cloud **do**

Remove all non–return points (NaN's) from the point cloud.

end for

Remove ground points:

for each point in the modified point cloud **do**

Remove all points with Z–values below T_G from point cloud.

end for

Create threshold point clouds:

Set $P_{HT} = \varnothing$.

Set $P_{LT} = \varnothing$.

for e **do**ach point P_j in the modified point cloud

if Point P_j has intensity $\geq T_H$ **then**

Add P_j to P_{HT}.

end if

if Point P_j has intensity $\geq T_L$ **then**

Add P_j to P_{LT}.

end if

end for

Extract features:

Extract features **f** using Algorithm 6.

Algorithm 6 LiDAR feature extraction.

input: LiDAR high–intensity point cloud $P_{HT} = \{x_j, y_j, z_j, i_j, r_j\}$.

input: LiDAR low–intensity point cloud $P_{LT} = \{x_j, y_j, z_j, i_j, r_j\}$.

input: Inner region x-extent: Δx_I (meters).

input: Inner region y-extent: Δy_I (meters).

input: Outer region x-extent: Δx_O (meters).

input: Outer region y-extent: Δy_O (meters).

input: LiDAR height above ground: $Z_L = 1.4$ (meters).

output: Feature vector **f**.

Cluster the high–intensity point cloud:

for e **do**ach point in **p** in the high–intensity point cloud

Cluster points and determine cluster centers.

end for

Calculate features:

for each cluster center point $c = (x_C, y_C, z_C)$ in the point cloud **do**

Determine all points in P_{HT} in the inner region using Equation 6.1 and calculate features 1, 13 and 17 from Table 6.1.

Determine all points in P_{HT} in the outer region using Equation 6.2 and calculate feature 4 from Table 6.1.

Determine all points in P_{LT} in the inner region using Equation 6.1 and calculate features 6, 7, 9, 10, 11, 14, 16 and 18 and 10 from Table 6.1.

Determine all points in P_{LT} in the outer region using Equation 6.2 and calculate features 2, 3, 5, 8, 12, 15, 19 and 20 from Table 6.1.

end for

Return $\mathbf{f} = [f_1, f_2, f_3, \cdots, f_{20}]$.

Table 6.1 describes the extracted features. For example, feature 1 is generated from the high threshold data, which consists solely of high–intensity points. The data is analyzed only for high–intensity points in the inner analysis region. This feature simply computes

the Z (height) extent, that is, the maximum Z value minus the minimum Z value. Note that some of the features are only processed on certain beams, such as feature 2. The features are listed in order of their discriminate power in descending order, e.g. feature 1 is the most discriminate, feature 2 the next most discriminate, etc. Initially, it was unknown how effective the features would be. We started with 134 features, and these were ranked in order of their discriminating power using the score defined as

$$SCORE = 500\frac{TP}{TP + FN} + 500\frac{TN}{TN + FP} \tag{6.3}$$

where TP, TN, FP and FN are the number of true positives, true negatives, false positive and false negatives, respectively [57]. A higher score is better, and scores range from 0 to 1,000. This score was used since training with overall accuracy tended to highly favor one class and provided poor generalizability, which may be a peculiarity of the chosen features and the training set. Figure 6.12 shows the score values versus the number of features (where the features are sorted in descending score order. The operating point is shown for $M = 20$ features (circles in the plot). The features generalize fairly well since the testing curve is similar to the training curve. In our system, using 20 features provided a good balance of performance versus computational complexity to compute the features. Each LiDAR scan (5 Hz scan rate) requires calculating features for each cluster. Experimentation showed that 20 features was near an upper limit of what the processor could calculate and not degrade the LiDAR processing frame rate.

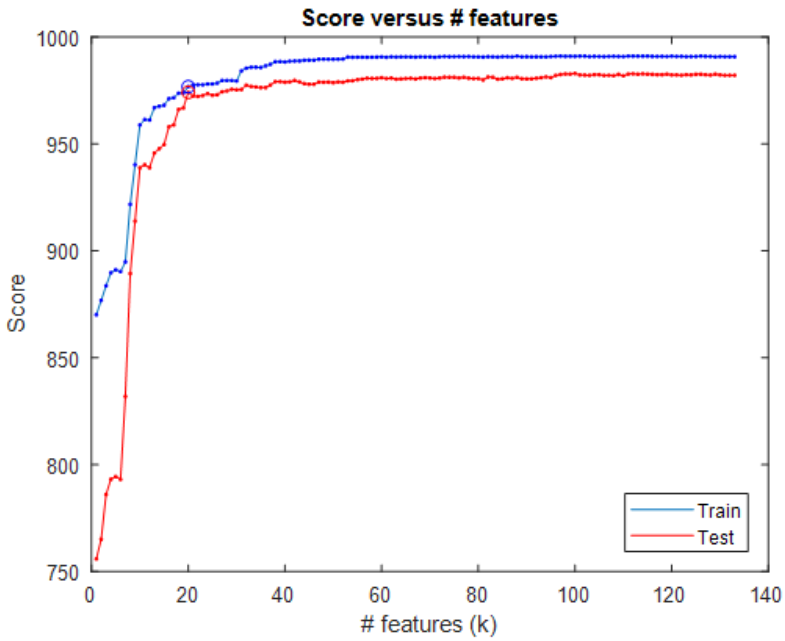


Figure 6.12

Score value versus number of concatenated features.

Table 6.1

Feature Descriptions. Data Subsets: HT = High threshold data, LT = Low threshold data. Extent $\{X\}$ = max$\{X\}$-min$\{X\}$. Extent $\{Y\}$ = max$\{Y\}$-min$\{Y\}$. Extent $\{Z\}$ = max$\{Z\}$-min$\{Z\}$. The LiDAR rings are shown in Figure 6.10.

Feature Number	**Data Subset**	**Analysis Region**	**Description**
1	HT	Inner	Extent of Z.
2	LT	Outer	Max of X and Y extents, beam 7. This is max{extent$\{X\}$,extent$\{Y\}$}.
3	LT	Outer	Max of X and Y extents, beam 5. This is max{extent$\{X\}$,extent$\{Y\}$}.
4	HT	Outer	Max$\{Z$ - LiDAR height$\}$.
5	LT	Outer	Extent of Z.
6	LT	Inner	Number of valid points in beam 7.
7	LT	Inner	Max of X and Y extents, beam 6. This is max{extent$\{X\}$,extent$\{Y\}$}.
8	LT	Outer	Number of valid points in beam 5.
9	LT	Inner	Extent of X.
10	LT	Inner	Number of points in beam 4.

6.3.6.4 SVM LiDAR beacon detection

To detect the beacons in the 3D LiDAR point cloud, we use a linear SVM [98] which makes a decision based on a learned linear discriminant rule operating on the hand-crafted features we have developed for differentiating beacons. Herein, liblinear was used to train the SVM [24]. A linear SVM operates by finding the optimal linear combination of features that best separates the training data. If the M testing features for a testing instance are given by $\mathbf{f} = [f_1, f_2, \cdots, f_M]^T$, where the superscript T is a vector transpose operator, then the SVM evaluates the discriminant

$$D = \mathbf{f} \cdot \mathbf{w} + b, \tag{6.4}$$

where **w** is the SVM weight vector and b is the SVM bias term. Herein, M was chosen to be 20. Equation 6.4 applies a linear weight to each feature and adds a bias term. The discriminant is optimized during SVM training. The SVM optimizes the individual weights in the weight vector to maximize the margin and provide the best overall discriminant capability of the SVM. The bias term lets the optimal separating hyperplane drift from the origin. Figure 6.13 shows an example case with two features. In this case, the $D = 0$ hyperplane is the decision boundary. The $D = +1$ and $D = -1$ hyperplanes are determined by the support vectors. The SVM only uses the support vectors to define the optimal boundary.

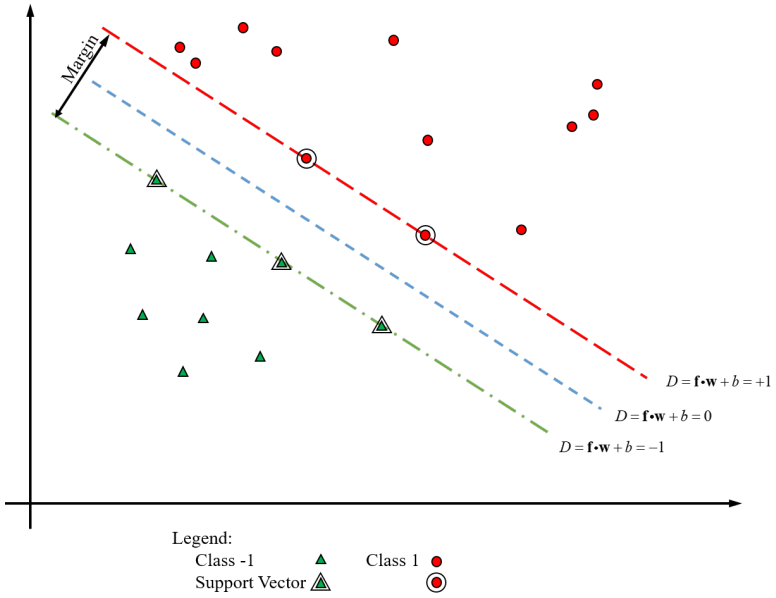


Figure 6.13

SVM 2D Example. The margin is the distance from the $D = -1$ to $D = 1$ hyperplanes.

The SVM was trained with beacon and non-beacon data extracted over multiple data collections. There were 13,190 beacon training instances and 15,209 non-beacon training instances. The test data had 12,084 beacons and 5,666 non-beacon instances.

In this work, $M = 20$ features were chosen. Each feature is first normalized by subtracting the mean of the feature values from the training data and dividing each feature by four times the standard deviation plus 10^{-5} (in order to prevent division by very small numbers). Subtracting the mean value centers the feature probability distribution function (PDF) around zero. Dividing by four times the standard deviation maps almost all feature values into the range $[-1, 1]$, because most of the values lie within $\pm 4\sigma_k$. The normalized features are calculated using

$$f'_k = \frac{f_k - \mu_k}{4\sigma_k + 10^{-5}} \tag{6.5}$$

where μ_k is the mean value of feature k and σ_k is the standard deviation of feature k for $k = 1, 2, \cdots, M$. The mean and standard deviation are estimated from the training data and then applied to the test data. The final discriminant is computed using 6.4 where the feature vector is the normalized feature vector.

The optimal feature weighs are shown in Figure 6.14 . Since all the features are normalized, features 1, 3, and 8 have the most influence on the final discriminant function.

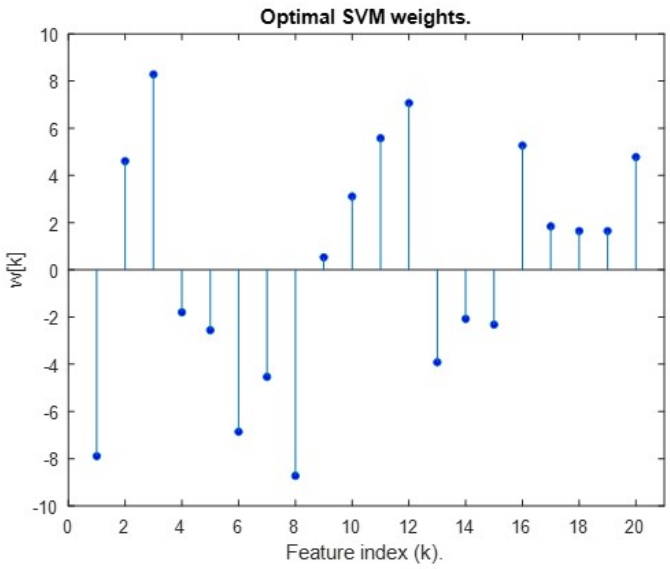


Figure 6.14

Optimal feature weights chosen by the SVM training optimizer.

If the discriminant $D \leq 0$, then the object is declared a beacon. Otherwise, it is declared a non–beacon (and ignored by the LiDAR beacon detection system). Once the SVM is trained, implementing the discriminant given in Equation 6.4 is trivial and uses minimal processing time.

Figure 6.15 shows the probability distribution function (PDF) of the LiDAR discriminant values for beacons and non–beacons, respectively. The PDFs are nearly linearly separable. The more negative the discriminant values, the more the object is beacon–like.

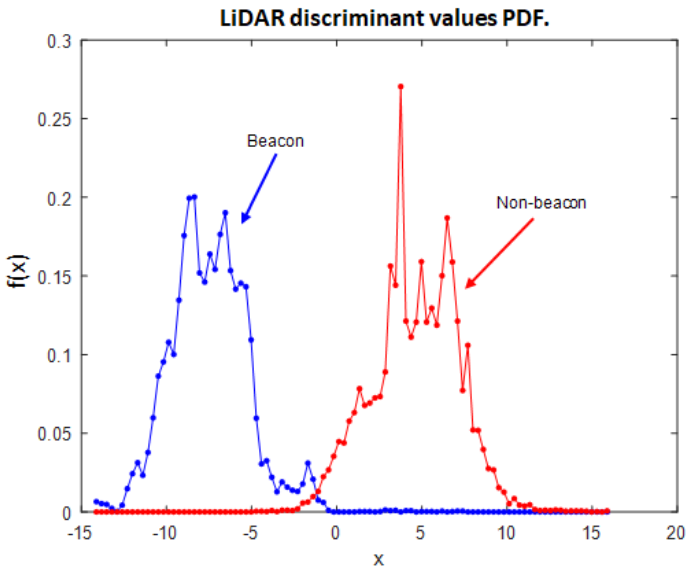


Figure 6.15

LiDAR discriminant values.

6.3.7 Mapping

This section discusses the camera detection mapping from bounding box coordinates to distance and angle. It also discusses the mapping of LiDAR discriminate value from range $[-\infty, \infty]$ to $[0, 1]$.

6.3.7.1 Camera detection mapping

The LiDAR and the camera both provide information on object detection, although in entirely different coordinate spaces. The LiDAR provides accurate angle and range estimates with the discriminant value offering a rough metric of confidence. The camera processing algorithms return bounding box coordinates, class labels, and confidence values in the range $[0, 1]$. However, effective fusion requires a common coordinate space. There-

fore, a mapping function is required to merge these sensor measurements into a shared coordinate space. To build this mapping function, the camera and LiDAR were mounted to the industrial vehicle and the LiDAR's accurate position measurements, alongside the camera's bounding boxes, were used to collect training data. These data were then used to train a neural network. The end result was a neural network that could project the camera's bounding boxes into a range and angle estimate in the LiDAR's coordinate frame, as seen in Figure 6.16. In this figure, the middle portion simply represents the bounding boxes from the camera system and the smaller boxes before the mapped results represents the NN mapper.

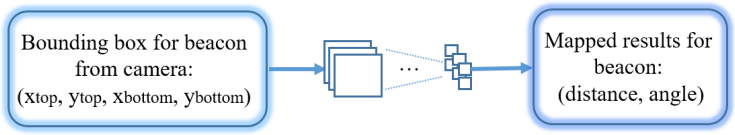


Figure 6.16

Detection mapping for camera.

The detailed NN structure is shown in Figure 6.17. It consists of ten fully connected layers: eight layers of ten neurons and two layers of twenty neurons. We choose this structure because of its simplicity to compute while outperforming other linear and non-linear regression methods.

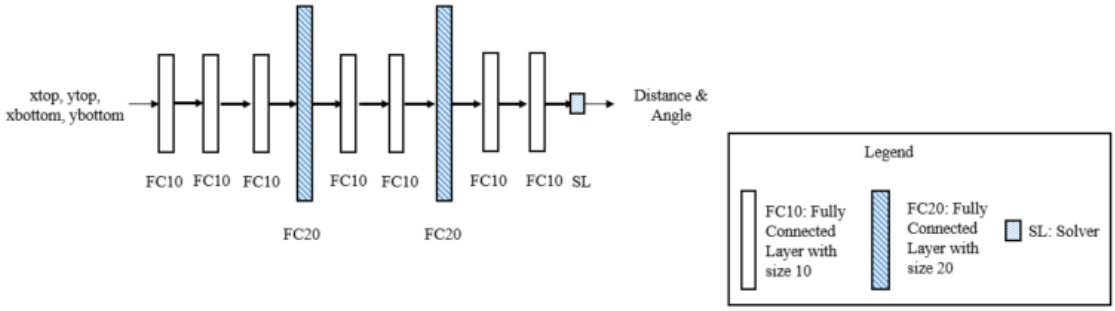


Figure 6.17

NN structure for detection mapping for camera.

6.3.7.2 LiDAR discriminate value mapping

With the camera bounding boxes mapped to LiDAR range angle, the first component of the mapping function is complete. However, there still remains the mapping of the discriminate and confidence values into a shared space. For this, the $[0, 1]$ range of the camera confidence offers a more intuitive measure than the LiDAR's $[-\infty, \infty]$ range for the discriminate value. Thus, the mapping function for this component will map the LiDAR discriminate to a $[0, 1]$ space for fusion with the camera. To do this, a logistic sigmoid of the form

$$f(x) = \frac{1}{1 + e^{-\alpha * x}} \tag{6.6}$$

was used. The discriminate value is multiplied by a gain term, α. The end result is a function that maps the discriminate value to a pseudo-confidence score with a range similar enough to the camera's confidence score for effective fusion.

6.3.8 Detection fusion

In this section, we discuss the fusion algorithm and hyperparameter optimization.

6.3.8.1 Fusion algorithm

With these two mapping components, a full pipeline can now be established to combine the LiDAR and camera data into forms similar enough for effective fusion. Using distance and angle information from both camera and LiDAR, we can correlate LiDAR and camera detections together. Their confidence scores can then be fused as shown in Algorithm 7.

Algorithm 7 Fusion of LiDAR and Camera detection.

input: Detection from LiDAR in the form of [distance, angle, pseudo-confidence score].

input: Detection from camera in the form of [distance, angle, confidence score].

input: Angular Threshold: A for angle difference between fused camera and LiDAR detection.

input: Confidence Threshold: C for final detection confidence.

output: Fused detection in the form of [distance, angle, detection confidence].

for each detection from camera **do**

if a corresponding LiDAR detection, with angle difference less than A, exists **then**

Fuse by using the distance and angle from LiDAR, and combine confidence scores

using Fuzzy logic to determine the fused detection confidence.

else

Create a new detection from camera, using the camera confidence score as the final detection confidence.

end if

end for

for each LiDAR detection that does not have a corresponding camera detection **do**

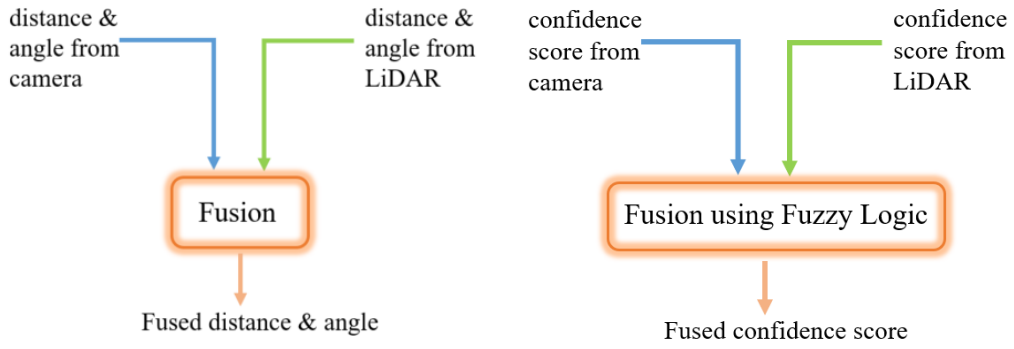
Create a new detection from LiDAR, using the LiDAR confidence score as the final detection confidence.

end for

for each fused detection **do** Remove detections with final confidences below the C confidence threshold.

end for

Return fused detection results.



(a) Fusion of distances and angles. (b) Fusion of confidence scores.

Figure 6.18

Fusion of distances, angles and confidence scores from camera and LiDAR.

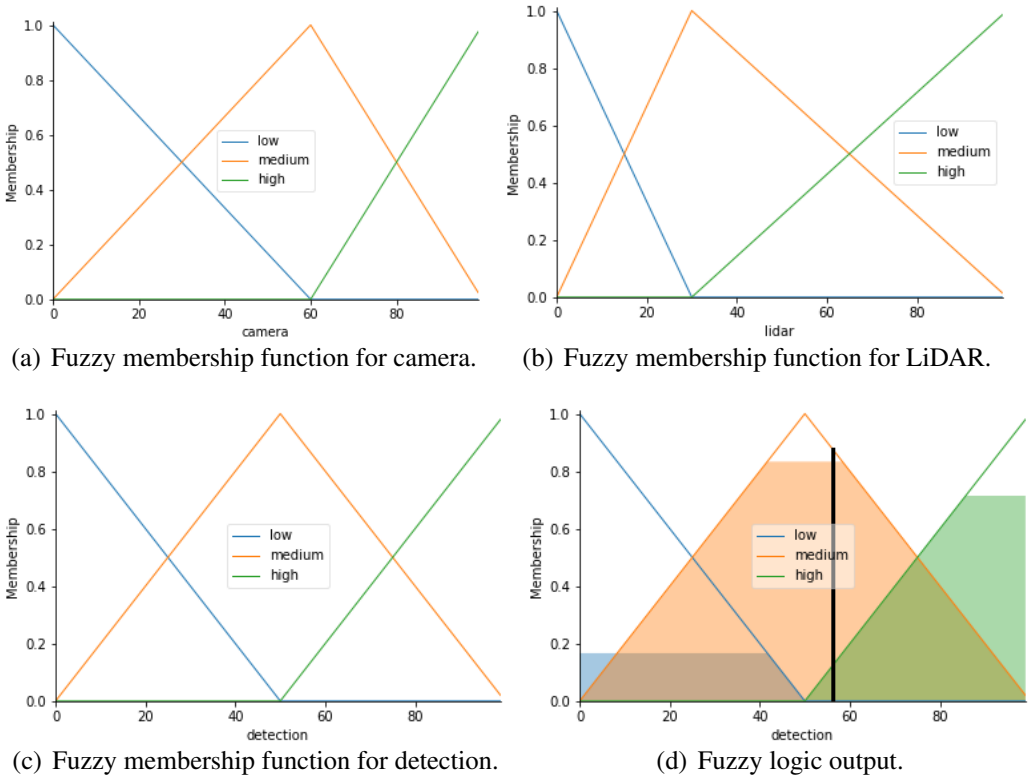
In this algorithm, the strategy for fusion of the distance and angle estimates is different from the fusion of confidence scores, which is shown in Figure 6.18. In the process of distance and angle fusion, for each camera detection, when there is a corresponding detection from LiDAR, we will use the distance and angle information from LiDAR, as LiDAR can provides much more accurate position estimation. When there is no corresponding LiDAR detection, we will use the distance, angle and confidence score from camera to create a new detection. For each detection from LiDAR that does not have corresponding camera detection, we will use the distance, angle and confidence score from LiDAR as final result. For confidence score fusion, we use fuzzy logic to produce a final confidence score when there is corresponding camera and LiDAR detection. The fuzzy rules used in the fuzzy logic system are as follows:

1. If the LiDAR confidence score is high, then the probability for detection is high.
2. If the camera confidence score is high, then the probability for detection is high.

3. If the LiDAR confidence score is medium, then the probability for detection is medium.
4. If the LiDAR confidence score is low and the camera confidence score is medium, then the probability for detection is medium.
5. If the LiDAR confidence score is low and the camera confidence score is low, then the probability for detection is low.

The fuzzy rules are chosen in this way because of the following reasons: First, if either the LiDAR or the camera shows a high confidence in their detection, then the detections are very likely to be true. This is the reason for Rules 1 and 2. Second, the LiDAR is robust at detection up to about 20 meters. Within that range, the confidence score from the LiDAR is usually high or medium. Beyond that range, the value will drop to low. This is the reason for Rule 3. Third, beyond the 20 meters detection range, the system will solely rely on the camera for detection. This is the reason for Rules 4 and 5.

Figure 6.19 shows the process of using fuzzy logic to combine the confidence scores from camera and LiDAR. The camera input, LiDAR input and detection output is fuzzified into fuzzy membership functions as shown in Figure 6.3.8.1, 6.3.8.1, and 6.3.8.1. The output is shown in Figure 6.3.8.1. In this example, when confidence score from the LiDAR is 80 and the score from the camera is 20, by applying the five pre–defined fuzzy rules, the output detection score is around 56.



(a) Fuzzy membership function for camera.

(b) Fuzzy membership function for LiDAR.

(c) Fuzzy membership function for detection.

(d) Fuzzy logic output.

Figure 6.19

Combination of confidence scores from camera and LiDAR using fuzzy logic. Figure best viewed in color.

6.3.8.2 Hyperparameter optimization

In machine learning, the value of hyperparameter is set before the learning process. In Equation 6.6, α controls the LiDAR sigmoid–like squashing function, and is one of the hyperparameters to be decided. In Algorithm 7, there is another hyperparameter threshold C for the final detection. Figure 6.20 shows where these two hyperparameters affect in the fusion process. For optimization of hyperparameters, some commonly used approaches include marginal likelihood [111], grid search [11, 43], evolutionary optimization [79], and many others. Herein, we utilized grid search. This method is widely used in machine

learning system as shown in [11, 43]. In grid search, the two hyperparameters are investigated by selecting candidate values for each, then generating a 2D matrix of their cross product.

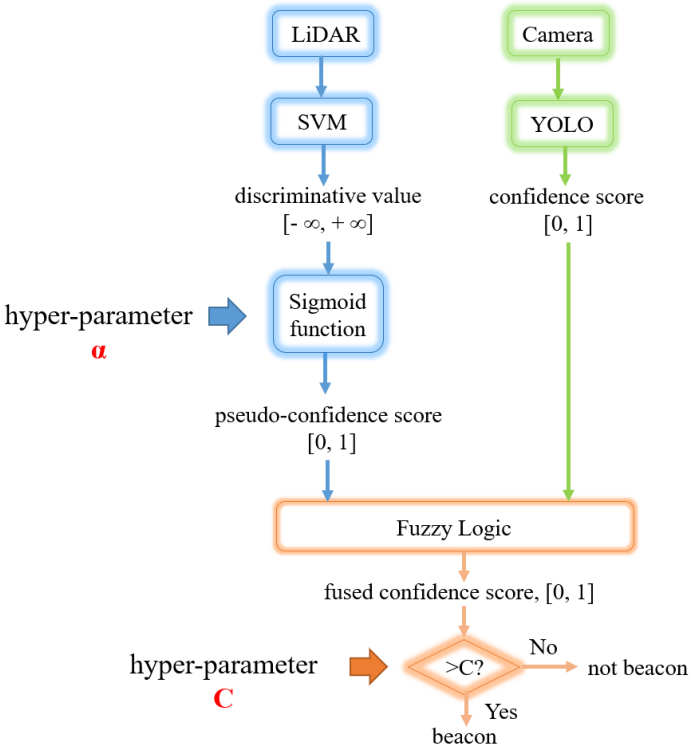


Figure 6.20

Fusion of confidence scores with two hyperparameters.

As we have two hyperparameters α and C to be tuned, we use Kolmogorov–Smirnov Test [9], also known as KS–Test, to determine which hyperparameter values to choose. In our application, KS–test is to find the α and C that maximize the value of TPR minus FPR. As we want to maximize TPR and minimize FPR, the purpose of KS–test is to find the balance between the two metrics.

The values chosen for α are $\frac{1}{100}$, $\frac{1}{500}$, $\frac{1}{1{,}000}$, $\frac{1}{5{,}000}$, $\frac{1}{10{,}000}$, $\frac{1}{50{,}000}$, $\frac{1}{100{,}000}$, $\frac{1}{500{,}000}$, and $\frac{1}{1{,}000{,}000}$, while the values for C are 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, and 0.95.We use 3641 testing LiDAR and camera pairs, and the resulted heat maps for TPR, FPR and KS–test results are shown in Figure 6.21. As we want the TPR to be high, in Figure 6.3.8.2, the green area shows where the highest TPR is, while the red are is where the lowest TPR is. Also, we want the FPR to be low, in Figure 6.3.8.2, the lower values are in the green area, while the higher values are in red area. To balance between TPR and FPR, Figure 6.3.8.2 shows the KS–test results, in other words the difference between TPR and FPR. We want to choose the α and C related to higher values (green area) in this heat map. Based on KS–test, there are three pairs of α and C values can be chosen and we choose α to be $\frac{1}{500{,}000}$ and C to be 0.65 in our experiment.

α \ C	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
1/100	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1/500	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.986
1/1,000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999	0.966	0.880
1/5,000	1.000	1.000	1.000	1.000	0.999	0.989	0.977	0.944	0.798	0.711
1/10,000	1.000	1.000	1.000	1.000	0.991	0.988	0.966	0.934	0.744	0.711
1/50,000	1.000	1.000	1.000	0.999	0.991	0.985	0.957	0.916	0.710	0.085
1/100,000	1.000	1.000	1.000	0.999	0.991	0.985	0.954	0.883	0.268	0.000
1/500,000	1.000	1.000	1.000	0.999	0.991	0.967	0.848	0.502	0.029	0.000
1/1,000,000	1.000	1.000	1.000	0.999	0.991	0.947	0.833	0.476	0.029	0.000

(a) True Positive Rate (TPR) changing with α and C.

α \ C	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
1/100	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.053	0.052	0.052
1/500	0.053	0.053	0.052	0.052	0.051	0.051	0.051	0.051	0.051	0.048
1/1,000	0.053	0.052	0.051	0.051	0.051	0.051	0.048	0.047	0.045	0.043
1/5,000	0.053	0.051	0.045	0.043	0.042	0.041	0.039	0.036	0.033	0.030
1/10,000	0.053	0.045	0.043	0.040	0.036	0.033	0.032	0.029	0.027	0.020
1/50,000	0.053	0.034	0.027	0.018	0.011	0.006	0.003	0.001	0.000	0.000
1/100,000	0.053	0.027	0.012	0.004	0.002	0.000	0.000	0.000	0.000	0.000
1/500,000	0.053	0.010	0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1/1,000,000	0.053	0.004	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000

(b) False Positive Rate (FPR) changing with α and C.

α \ C	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95
1/100	0.947	0.947	0.947	0.947	0.947	0.947	0.947	0.947	0.948	0.948
1/500	0.947	0.947	0.948	0.948	0.949	0.949	0.949	0.949	0.949	0.938
1/1,000	0.947	0.948	0.949	0.949	0.949	0.949	0.952	0.952	0.921	0.837
1/5,000	0.947	0.949	0.955	0.957	0.957	0.948	0.938	0.908	0.765	0.681
1/10,000	0.947	0.955	0.957	0.960	0.955	0.955	0.934	0.905	0.717	0.691
1/50,000	0.947	0.966	0.973	0.981	0.980	0.979	0.954	0.915	0.710	0.085
1/100,000	0.947	0.973	0.988	0.995	0.989	0.985	0.954	0.883	0.268	0.000
1/500,000	0.947	0.990	0.997	0.999	0.991	0.967	0.848	0.502	0.029	0.000
1/1,000,000	0.947	0.996	0.999	0.999	0.991	0.947	0.833	0.476	0.029	0.000

(c) KS–test results changing with α and C.

Figure 6.21

Heat maps demonstrating the TPR, FPR and KS–test results changing with α and C. best viewed in color.

6.4 Experiment

In our experiments, the LiDAR we use is a Quanergy M8-1 LiDAR shown in Figure 6.4. It has eight beams with a vertical spacing of approximately 3°. The camera images

are collected by a FLIR Chameleon3 USB camera with Fujinon 6mm lens shown in Figure 6.4.

(a) Quanergy M8-1 LiDAR. (b) FLIR Chameleon3 USB camera with Fujinon 6mm lens.

Figure 6.22

Sensors used in our experiments.

6.4.1 Data collections summary

To collect data and test our system in a relevant environment, we needed an open outdoor space without much clutter and interference from random objects. We chose an industrial setting which is very similar to the scenario where we will implement our system. The location has an office building and a large, relatively flat concrete area adjacent to the building. In the environment, there are a few parked cars, a trailer, a couple of gas storage tanks, etc.

Twenty extensive data collections were conducted from August 2017 to March 2018. These tests are designed to collect training data for our system and testing data to check the system's performance and to tune the system parameters. Many tests consisted of

beacons placed in a random position in the sensors' FOV. Other tests had beacons in known locations in order to assess the LiDAR and camera accuracies for estimating the range and distance. Later tests used the LiDAR as the "ground truth", since it is highly accurate at estimating range and distance.

Most of the initial tests had the industrial vehicle stationary. Later, we performed tests with the industrial vehicle moving. For the initial tests where we needed to know beacon locations, we marked the ground different distances and angles. The ground point straight below the LiDAR is used as the 0 meter distance and 0° angle is directly in front of the industrial vehicle. From there we marked the distance from 3 meters at each meter interval up to 40 meters across the 0° angle line. The 0° angle is directly in front of the industrial vehicle. We also marked the $-20^\circ, -15^\circ, -10^\circ, -5^\circ, 0^\circ, 5^\circ, 10^\circ, 15^\circ$ and 20° lines for beacon location. In this way, we constructed a dataset of beacons with labeled ground truths. For static beacon testing, we placed the beacon in both known and unknown locations and recorded data.

For the stationary vehicle testing, we set up several tests to evaluate different parts of the system. For instance, a beacon was placed on a very short wooden dolly attached to a long cord. This allowed the beacon to be moved while the people moving it were not in the sensors' fields of view. This allowed the beacon to be placed at a known distance along the 0° angle line and drug toward the vehicle to collect data at continuous locations. We also did similar tests along the $-40^\circ, -20^\circ, -10^\circ, 10^\circ, 20^\circ$ and 40° angle lines. Finally, we collected data where the beacon was drug from the -40° angle to all the way across the

40° angle at roughly fixed distances. These data collections gave us a large amount of data to use in training and testing our system.

We also performed data collections with humans. The retro–reflective stripes on the vests present bright returns to the LiDAR, which may cause false positives. To measure these effects, we had different people wear a highly reflective vest and stand in front of the LiDAR and cameras in a known location. Data was gathered with the person in multiple orientations facing toward and away from the camera. This test was also performed without the reflective safety vests. Other tests had people walking around in random directions inside the sensors' FOV. The data collected here were part of the non–beacon data used to train the SVM.

In order to examine interference when people and beacons were also in the scene, we had people wearing a vest standing close to the beacon in known locations. Then we recorded the data which helped the system to distinguish the beacon and people even if in close contact. Furthermore, we also wanted some dynamic cases of beacons and people in close proximity. To gather this data, the beacon was pulled around using the rope and dolly while a second person moved around the beacon.

There were also a series of data collections with the vehicle driving straight at or just past a beacon or another object, to test that the sensor system was stable under operating conditions. The vehicle was driven at a constant speed, accelerating and braking (to make the vehicle rock), and in a serpentine manner. Finally, we performed many data collections with the vehicle driving around with different obstacles and beacons present. This was mainly to support system integration and tune the MPC controller and actuators. All

of these collections provided a rich dataset for system analysis, tuning, and performance evaluation.

6.4.2 Camera detection training

Training images for YOLO also include images taken by a Nikon D7000 camera and images from PASCAL VOC dataset [22]. The total number of training images is around $25,000$.

The network structure of YOLO is basically the same as the default setting. One difference is the number of filters in the last layer. This number is related to the number of object categories we are trying to detect and the number of base bounding boxes for each of the grid sections that YOLO divides the image into. For example, we can divide each image into 13 by 13 grids. For each grid, YOLO predicts 5 bounding boxes, and for each bounding box, there are $5 + N$ parameters (N represents the number of categories to be predicted). As a result, the last layer filter size is $13 \times 13 \times 5 \times (5 + N)$. Another difference is that we change the learning rate to 10^{-5} to avoid divergence.

6.4.3 LiDAR detection training

The LiDAR data were processed through a series of steps. We implemented a ground clutter removal algorithm to keep ground points from giving us false alarms, to avoid high reflections from reflective paints on the ground, and to reduce the size of our point cloud. Clusters of points are grouped into distinct objects and are classified using a SVM. The SVM is trained using a large set of data which has beacons and non–beacon objects. The LiDAR data were divided into two disjoint sets: training and testing. In the training dataset,

there were 13,190 beacons and 15,209 non–beacons. The testing dataset contained 5,666 beacons and 12,084 non–beacons. The confusion matrices for training and testing are shown in Tables 6.2 and 6.3. The results show good performance for the training and testing datasets. Both datasets show over 99.7% true positives (beacons) and around 93% true negatives (non–beacons).

Table 6.2

LiDAR training confusion matrix.

	Beacon	Non–Beacon
Beacon	13,158	32
Non–Beacon	1,144	14,056

Table 6.3

LiDAR testing confusion matrix.

	Beacon	Non–Beacon
Beacon	5,653	13
Non–Beacon	780	11,304

6.5 Results and Discussion

6.5.1 Camera detection mapping results

Camera and LiDAR detections generate both different types of data and data on different scales. The LiDAR generates range and angle estimates, while the camera generates bounding boxes. The LiDAR discriminant score can be any real number, while the camera confidence score is a real number in the range $[0, 1]$. The data from camera detection is first mapped into the form of distance and range, the same form as the data reported from Li-

DAR. We can accomplish this conversion fairly accurately because we know the true size of the beacon. The data collected for mapping covers the entire horizontal FOV of camera, which is -20° to 20° relative to the front of the industrial vehicle. The distance covered is from 3 to 40 meters, which is beyond the maximum detection range of our LiDAR detection algorithm. We collected around 3,000 pairs of camera and LiDAR detection for training of the mapping NN, and 400 pairs for testing.

From Figure 6.23, we observe that the larger the x-coordinate, the larger the angle from the camera, and the larger the size of the bounding box, the nearer the object to the camera. From our training data, we see that the bounding box's x-coordinate has an approximately linear relationship with the LiDAR detection angle, as shown in Figure 6.5.1. We also observe that the size of the bounding box from the camera detection has a nearly negative exponential relationship with the distance from LiDAR detection as shown in Figure 6.5.1. To test these relationships, we ran least squares fitting algorithms to estimate coefficients for a linear fit between the [$Xmin$, $Xmax$] of the camera bounding box and the LiDAR angle, and for an exponential fit between the [$Width$, $Height$] and the LiDAR estimated distance.

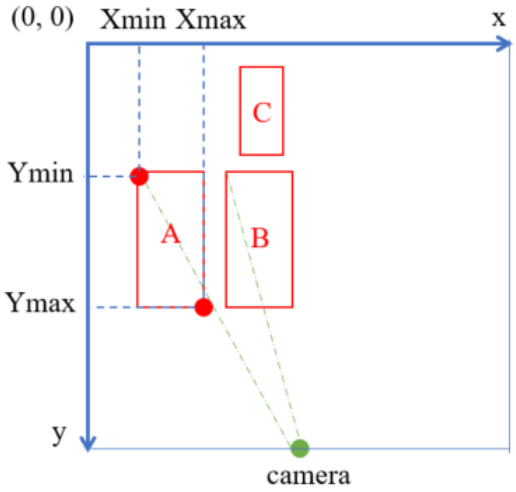
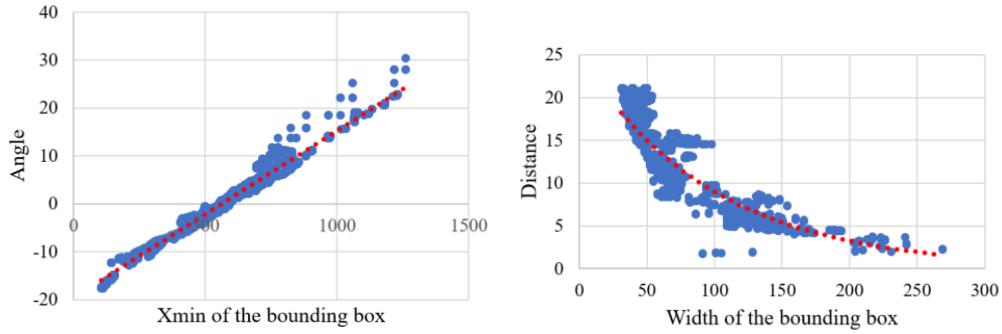


Figure 6.23

One sample frame with three different bounding boxes.



(a) Bounding box $Xmin$ from camera vs. angle from LiDAR. Xmin units are pixels and angles units are degrees.

(b) Bounding box $Xmin$ from camera vs. angle from LiDAR. Bounding box width is pixels, and distance is meters.

Figure 6.24

Scatter plots of camera/LiDAR training data for fusion. (a) Camera bounding box $Xmin$ vs. LiDAR estimated angle. (b) Camera bounding box width vs. LiDAR estimated distance.

Herein, we compare the NN and linear regression results methods for angle estimation, and we also compare the results using NN and exponential curve fitting method for distance. The results are shown in Table 6.4. The table shows the mean squared error (MSE) using both the NN and the linear regression for the angle estimation, as well as the MSE for distance estimation. The table also reports the coefficient of determination, or r^2 scores [20]. The r^2 scores show a high measure of goodness of fit for the NN approximations. We find that the NN provides more accurate predictions on both distance and angle.

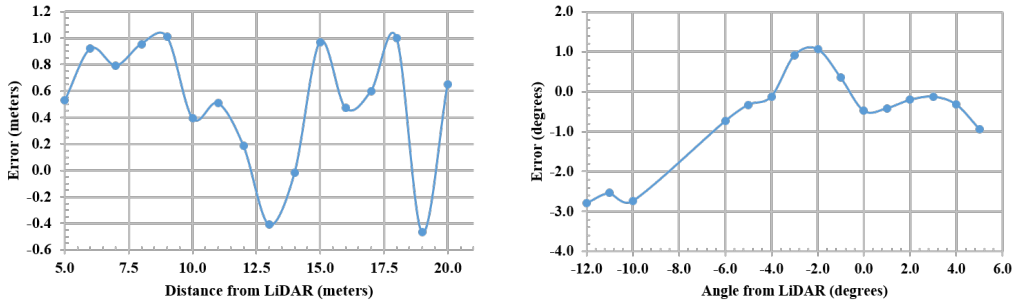
Table 6.4

Mapping analysis. Best results are shown in **bold**.

	Angle (degree)		Distance (meter)	
	NN	Linear regression	NN	Exponential curve fitting
Mean squared error	**0.0467**	0.0468	**0.0251**	0.6278
r^2 score	**0.9989**	**0.9989**	**0.9875**	0.9687

6.5.2 Camera detection results

In order to see how well the camera works on its own for reporting distance and angle for its detections, 400 testing images are processed. We compare the mapping results with the LiDAR detection, which is treated as ground–truth. The results are shown in Figure 6.25. From Figure 6.5.2, we can see that the camera can fairly accurately predict the angle with a maximum error around 3°, and this only happens near the image edges. On the other hand, the camera's distance prediction has relatively large errors, as shown in Figure 6.5.2. Its maximum error is around 1 meter, which happens when the distance is around 6 meters, 9 meters, 15 meters, and 18 meters.



(a) Angle error of camera.

(b) Distance error of camera.

Figure 6.25

Camera errors

6.5.3 Fusion results

We also compare the true positive detections, false positive detections and false negative detections using only LiDAR data with linear SVM method, only camera image results with YOLO, and the proposed fusion method. The results are shown in Tables 6.5 and 6.6.

Table 6.5

Fusion performance: 3–20 meter range.

	LiDAR only	Camera only	Fusion
True Positive Rate (TPR)	93.90 %	97.60 %	97.60 %
False Positive Rate (FPR)	27.00 %	0.00 %	6.69 %
False Negative Rate (FNR)	6.10 %	2.40 %	2.40 %

Table 6.6

Fusion performance: 20–40 meter range.

	Camera only	Fusion
True Positive Rate (TPR)	94.80 %	94.80 %
False Positive Rate (FPR)	0.00 %	0.00 %
False Negative Rate (FNR)	5.20 %	5.20 %

In Tables 6.5 and 6.6, TPR, FPR and FNR are calculated by Equations 6.7a, 6.7b, and 6.7c, respectively, in which TP is the number of true positives (the number of true beacons detected), TN is the number of true negatives (the number of non-beacons detected correctly), and FP is the number of false positive (there is an object detected as a beacon that is not a beacon), and FN is the number of false negatives (there is a beacon but we have no detection).

$$True\ Positive\ Rate\ (TPR) = \frac{TP}{TP + FN} \tag{6.7a}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{FP + TN} \tag{6.7b}$$

$$False\ Negative\ Rate\ (FNR) = \frac{FN}{TP + FN} \tag{6.7c}$$

From these results, one can see that from the camera there are more detections within its FOV up to 40 meters, while its estimation of position, especially for distance, is not accurate. The camera, however, can fail more quickly in rain or bad weather, so we can't always rely on the camera in bad weather. On the other hand, the LiDAR has accurate position estimation for each point, but the current LiDAR processing cannot reliably detect beyond about 20 meters, and its false detections (including false positives and false negatives) is relatively high. After fusion, the results show that it the system can detect beacons up to 40 meters like the camera can, and the LiDAR can help more accurately estimate the position of these detections, which is important to the control system (which will need different control responses when there is a beacon 20 meters away versus the case where

the beacon is 5 meters away). Figure 6.26 shows the benefit of using camera and LiDAR fusion system.

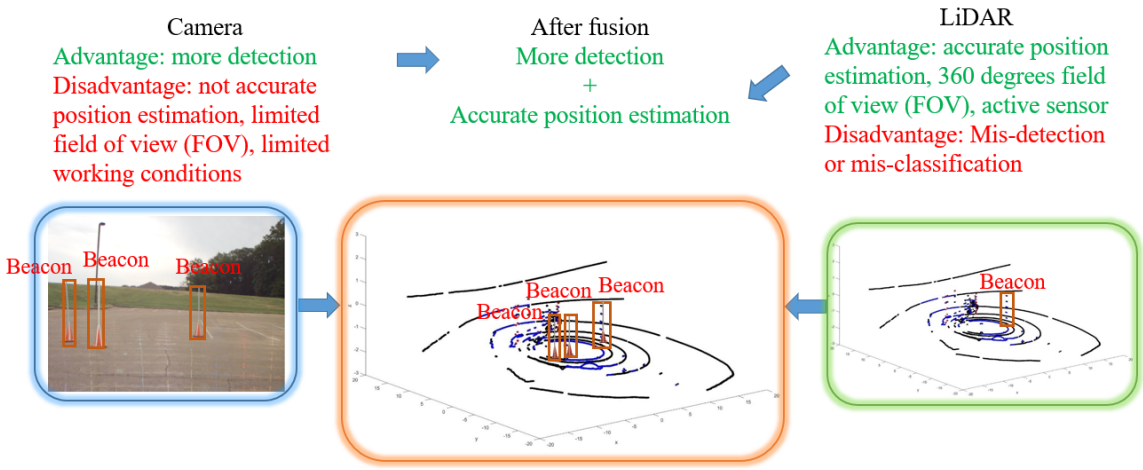


Figure 6.26

Benefit of using camera and LiDAR fusion system.

6.6 Conclusions

In this paper, we proposed a multi–sensor detection system that fuses camera and LiDAR detections for a more accurate and robust beacon detection system. The proposed system has been designed as a real–time industrial system for collision avoidance, and tested with prototypes in various scenarios. The results show that fusion helps to obtain more accurate position and label information for each detection. It also helps to create detection beyond the range of LiDAR while within the range of camera.

CHAPTER 7

CONCLUSION

For my research on fusion for object detection, I designed a computational intelligent fusion system for more accurate and robust detection. In my pursuit of this goal, I proposed a measure for evaluating different sources, applied this measure for interval-valued source fusion, and extended another measure for axis-aligned bounding box (AABB) fusion (i.e. detection fusion). I also proposed a multi–sensor fusion system, using LiDAR and camera. This system is applied in a real-time industrial multi-sensor collision avoidance system. In both systems, I utilized deep learning for object detection in images

Specifically, in Chapter 3, I propose a method to calculate measure of conflict, which I name conflict measure (CM). This method does not need external knowledge about the credibility of each source. Instead, it uses the information from the sources themselves to help assess the credibility of each source. The result shows that this measure can help identifying the most conflicting source among sensors. In Chapter 4, I apply the proposed CM to fuse four independent sources of tracking information from various stereo cameras. In this work, all sensors look at the same scene, but there is no external knowledge about the sensors other than the data they provide. We use CM to estimates the conflicts between sensors. CM is also used to estimate the credibility of each sensor. Based on these

credibility, we fuse tracking results from various sources together. The results using the proposed method are better than using average operation. In Chapter 5, I apply another type of measure (FM of agreement), and extended fuzzy integral (FI) to AABB, which is named AABBFI. By using AABBFI with FM of agreement, I propose a computational intelligence system for more accurate object detection in real-time. The proposed system uses online image augmentation before the detection stage during testing, and fuses the detection results after. The fusion method is computationally intelligent based on the dynamic analysis of agreement among inputs. Comparing with other fusion operations such as average, median and NMS, the proposed methods produces more accurate results in real-time. In Chapter 6, I propose a multi–sensor fusion system, which incorporates advantages and mitigate disadvantages of each type of sensor (LiDAR and camera). Generally, camera can provide more texture and color information, but it cannot work in low visibility. On the other hand, LiDAR can provide accurate point positions and work at night or in moderate fog or rain. The proposed system uses the advantages of both camera and LiDAR and mitigate their disadvantages. The results show that comparing with LiDAR or camera detection alone, the fused result can extend the detection range up to 40 meters with increased detection accuracy and robustness.

BIBLIOGRAPHY

[1] L. A. Alexandre, "3D object recognition using convolutional neural networks with transfer learning between input channels," *Intelligent Autonomous Systems 13*, Springer, 2016, pp. 889–898.

[2] B. Alrifaee, J. Maczijewski, and D. Abel, "Sequential Convex Programming MPC for Dynamic Vehicle Collision Avoidance," *IEEE Conference on Control TEchnology and Applications*. IEEE, 2017.

[3] D. T. Anderson, T. C. Havens, C. Wagner, J. M. Keller, M. F. Anderson, and D. J. Wescott, "Extension of the fuzzy integral for general fuzzy set-valued information," *IEEE Transactions on Fuzzy Systems*, vol. 22, no. 6, 2014, pp. 1625–1639.

[4] A. Baraka, G. Panoutsos, M. Mahfouf, and S. Cater, "A Shannon entropy-based conflict measure for enhancing granular computing-based information processing," *Granular Computing (GrC), 2014 IEEE International Conference on*, Oct 2014, pp. 13–18.

[5] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Improving object detection with one line of code," *arXiv preprint arXiv:1704.04503*, 2017.

[6] N. Brunel, V. Hakim, and M. J. Richardson, "Single neuron dynamics and computation," *Current opinion in neurobiology*, vol. 25, 2014, pp. 149–155.

[7] J. Castorena, U. S. Kamilov, and P. T. Boufounos, "Autocalibration of LIDAR and optical cameras via edge alignment," *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2862–2866.

[8] S.-H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *City*, vol. 1, no. 2, 2007, p. 1.

[9] I. M. Chakravarty, J. Roy, and R. G. Laha, *Handbook of methods of applied statistics*, McGraw-Hill, 1967.

[10] Y. Chang, A. Habib, D. Lee, and J. Yom, "Automatic classification of lidar data into ground and non-ground points," *International archives of Photogrammetry and Remote Sensing*, vol. 37, 2008, pp. 463–468.

[11] D. Chicco, "Ten quick tips for machine learning in computational biology," *BioData mining*, vol. 10, no. 1, 2017, p. 35.

[12] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[13] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, 2016, pp. 379–387.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005, vol. 1, pp. 886–893.

[15] C. Davenport, Y. Liu, H. Pan, J. Gafford, S. Abdelwahed, M. Mazzola, J. E. Ball, and R. F. Burch, "A kinematic modeling framework for prediction of instantaneous status of towing vehicle systems," *SAE International Journal of Passenger Cars-Mechanical Systems*, vol. [Accepted], 2018.

[16] L. C. de Barros, R. C. Bassanezi, and W. A. Lodwick, "The Extension Principle of Zadeh and Fuzzy Numbers," *A First Course in Fuzzy Logic, Fuzzy Dynamical Systems, and Biomathematics*, Springer, 2017, pp. 23–41.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[18] M.-M. Deza and E. Deza, *Dictionary of distances*, Elsevier, Amsterdam, Oxford, 2006.

[19] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, 2006, pp. 1–35.

[20] N. R. Draper and H. Smith, *Applied regression analysis*, vol. 326, John Wiley & Sons, Toronto, 2014.

[21] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise.," *Kdd*, 1996, vol. 96, pp. 226–231.

[22] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, 2010, pp. 303–338.

[23] M. Everingham and J. Winn, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit," *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 2011.

[24] R. Fan, K. Chang, and C. Hsieh, "LIBLINEAR: A library for large linear classification," *The Journal of Machine Learning*, vol. 9, 2008, pp. 1871–1874.

[25] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, 2010, pp. 1627–1645.

[26] A.-I. García-Moreno, J.-J. Gonzalez-Barbosa, F.-J. Ornelas-Rodriguez, J. B. Hurtado-Ramos, and M.-N. Primo-Fuentes, "LIDAR and panoramic camera extrinsic calibration approach using a pattern plane," *Mexican Conference on Pattern Recognition*. Springer, 2013, pp. 104–113.

[27] L. Garmendia, "The evolution of the concept of fuzzy measure," *Intelligent Data Mining*, Springer, Berlin, Heidelberg, 2005, pp. 185–200.

[28] S. Gidaris and N. Komodakis, "Locnet: Improving localization accuracy for object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 789–798.

[29] R. Giordano, "A fuzzy conflict measure for conflict dissolution in drought management," *Computational Intelligence for Measurement Systems and Applications (CIMSA), 2010 IEEE International Conference on*, Sept 2010, pp. 60–65.

[30] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[31] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2154–2161.

[32] X. Gong, Y. Lin, and J. Liu, "3D LIDAR-camera extrinsic calibration using an arbitrary trihedron," *Sensors*, vol. 13, no. 2, 2013, pp. 1902–1918.

[33] X. Gong, Y. Lin, and J. Liu, "Extrinsic calibration of a 3D LIDAR and a camera using a trihedron," *Optics and Lasers in Engineering*, vol. 51, no. 4, 2013, pp. 394–401.

[34] J. C. Gower, "A general coefficient of similarity and some of its properties," *Biometrics*, 1971, pp. 857–871.

[35] M. Grabisch, T. Murofushi, and M. Sugeno, *Fuzzy Measures and Integrals: Theory and Applications*, Physica-Verlag, New York, 2000.

[36] T. Havens, D. Anderson, and C. Wagner, "Constructing Meta-Measures From Data-Informed Fuzzy Measures for Fuzzy Integration of Interval Inputs and Fuzzy Number Inputs," *Fuzzy Systems, IEEE Transactions on*, November 2014.

[37] T. Havens, D. Anderson, C. Wagner, H. Deilamsalehy, and D. Wonnacott, "Fuzzy integrals of crowd-sourced intervals using a measure of generalized accord," *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*, July 2013, pp. 1–8.

[38] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," *arXiv preprint arXiv:1703.06870*, 2017.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[40] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, 1989, pp. 359–366.

[41] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," *arXiv preprint arXiv:1705.02950*, 2017.

[42] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *arXiv preprint arXiv:1312.5402*, 2013.

[43] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al., *A practical guide to support vector classification*, Tech. Rep., National Taiwan University, 2010.

[44] P. Hu and D. Ramanan, "Finding tiny faces," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 1522–1530.

[45] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get M for free," *arXiv preprint arXiv:1704.00109*, 2017.

[46] S. Iagnemma, KarlAnderson, S. Peters, and T. Pilutti, "An Optimal-control-based Framework for Trajectory Planning, Thread Assessment, and Semi-Autonomous Control of Passenger Vehicles in Hazard Avoidance Scenarios ," *International Journal of Vehicle Autonomous Systems*, vol. [8], no. 2,3,4, 2010, pp. 190–216.

[47] M. A. Islam, D. T. Anderson, A. J. Pinar, and T. C. Havens, "Data-Driven Compression and Efficient Learning of the Choquet Integral," *IEEE Transactions on Fuzzy Systems*, 2017.

[48] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, 1901, pp. 547–579.

[49] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-Normalizing Neural Networks," *arXiv preprint arXiv:1706.02515*, 2017.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[51] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995, pp. 255–258.

[52] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, 1989, pp. 541–551.

[53] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2278–2324.

[54] J. Levinson and S. Thrun, "Automatic Online Calibration of Cameras and Lasers.," *Robotics: Science and Systems*, 2013, vol. 2.

[55] I. B. Levitan and L. K. Kaczmarek, *The neuron: cell and molecular biology*, Oxford University Press, New York, 2015.

[56] J. Li, X. He, and J. Li, "2D LiDAR and camera fusion in 3D modeling of indoor environment," *Aerospace and Electronics Conference (NAECON), 2015 National*. IEEE, 2015, pp. 379–383.

[57] K. Lillywhite, D.-J. Lee, B. Tippetts, and J. Archibald, "A feature construction method for general object recognition," *Pattern Recognition*, vol. 46, no. 12, 2013, pp. 3300–3314.

[58] C.-H. Lin, J.-Y. Chen, P.-L. Su, and C.-H. Chen, "Eigen-feature analysis of weighted covariance matrices for LiDAR point cloud classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 94, 2014, pp. 70–79.

[59] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1925–1934.

[60] J. Liu, P. Jayakumar, J. Stein, and T. Ersal, "A Multi-stage Optimization Formulation for MPC-based Obstacle Avoidance in Autonomous Vehicles Using a LiDAR Sensor," *Proceedings of the ASME Dynamic Systems and Control Conference*. ASME, 2015, vol. [Accepted].

[61] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *European Conference on Computer Vision*. Springer, 2016, pp. 21–37.

[62] Y. Liu, C. Davenport, J. Gafford, M. Mazzola, J. Ball, S. Abdelwahed, M. Doude, and R. Burch, *Development of A Dynamic Modeling Framework to Predict Instantaneous Status of Towing Vehicle Systems*, Tech. Rep., SAE Technical Paper, 2017.

[63] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, 1982, pp. 129–137.

[64] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *Proc. ICML*, 2013, vol. 30.

[65] W. Maddern and P. Newman, "Real-time probabilistic fusion of sparse 3D LIDAR and dense stereo," *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2181–2188.

[66] T. Malisiewicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-svms for object detection and beyond," *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 89–96.

[67] R. Maree, P. Geurts, J. Piater, and L. Wehenkel, "Random subwindows for robust image classification," *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. IEEE, 2005, vol. 1, pp. 34–40.

[68] A. Martin, A.-L. Jousselme, and C. Osswald, "Conflict measure for the discounting operation on belief functions," *Information Fusion, 2008 11th International Conference on*, June 2008, pp. 1–8.

[69] A. Martin, A.-L. Jousselme, and C. Osswald, "Conflict measure for the discounting operation on belief functions," *Information Fusion, 2008 11th International Conference on*. IEEE, 2008, pp. 1–8.

[70] A. Mastin, J. Kepner, and J. Fisher, "Automatic registration of LIDAR and optical images of urban scenes," *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2639–2646.

[71] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[72] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, 1943, pp. 115–133.

[73] X. Meng, N. Currit, and K. Zhao, "Ground filtering algorithms for airborne LiDAR data: A review of critical issues," *Remote Sensing*, vol. 2, no. 3, 2010, pp. 833–860.

[74] K. Miadlicki, M. Pajor, and M. Saków, "Ground plane estimation from sparse LIDAR data for loader crane sensor fusion system," *Methods and Models in Automation and Robotics (MMAR), 2017 22nd International Conference on*. IEEE, 2017, pp. 717–722.

[75] M. Minsky and S. Papert, *Perceptrons.*, MIT press, Cambridge, Massachusetts and London, England, 1969.

[76] T. Murofushi and M. Sugeno, "An interpretation of fuzzy measures and the Choquet integral as an integral with respect to a fuzzy measure," *Fuzzy sets and Systems*, vol. 29, no. 2, 1989, pp. 201–227.

[77] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[78] A. Napier, P. Corke, and P. Newman, "Cross-calibration of push-broom 2d lidars and cameras in natural scenes," *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3679–3684.

[79] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, J. H. Moore, et al., "Automating biomedical data science through tree-based pipeline optimization," *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 123–137.

[80] G. Pandey, J. R. McBride, S. Savarese, and R. M. Eustice, "Automatic extrinsic calibration of vision and lidar by maximizing mutual information," *Journal of Field Robotics*, vol. 32, no. 5, 2015, pp. 696–722.

[81] Y. Park, S. Yun, C. S. Won, K. Cho, K. Um, and S. Sim, "Calibration between color camera and 3D LIDAR instruments with a polygonal planar board," *Sensors*, vol. 14, no. 3, 2014, pp. 5333–5353.

[82] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[83] G. Rafiee, S. S. Dlay, and W. L. Woo, "Region-of-interest extraction in low depth of field images using ensemble clustering and difference of Gaussian approaches," *Pattern Recognition*, vol. 46, no. 10, 2013, pp. 2685–2699.

[84] P. Rashidi and H. Rastiveis, "Ground Filtering LiDAR Data Based on Multi-Scale Analysis of Height Difference Threshold," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2017, pp. 225–229.

[85] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[86] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[87] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, 2015, pp. 91–99.

[88] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*, Cornell Aeronautical Laboratory, 1957.

[89] T. J. Ross, *Fuzzy logic with engineering applications*, John Wiley & Sons, Chennai, India, 2010.

[90] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, 1986, pp. 533–536.

[91] L. Rummelhard, A. Paigwar, A. Nègre, and C. Laugier, "Ground estimation and point cloud segmentation using SpatioTemporal Conditional Random Field," *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 1105–1110.

[92] D. Silver et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, 2017, pp. 354–359.

[93] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016, pp. 484–489.

[94] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[95] M. Sugeno, "Theory of fuzzy integrals and its applications," *Doctorial Thesis*, 1974.

[96] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[97] C. Szegedy, S. Reed, D. Erhan, D. Anguelov, and S. Ioffe, "Scalable, high-quality object detection," *arXiv preprint arXiv:1412.1441*, 2014.

[98] V. Vapnik, *Statistical Learning Theory*, Wiley, New York, 1998.

[99] F. Vasconcelos, J. P. Barreto, and U. Nunes, "A minimal solution for the extrinsic calibration of a camera and a laser-rangefinder," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, 2012, pp. 2097–2107.

[100] C. Wagner and D. Anderson, "Extracting meta-measures from data for fuzzy aggregation of crowd sourced information," *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*, June 2012, pp. 1–8.

[101] L. Wang and Y. Zhang, "LiDAR Ground Filtering Algorithm for Urban Areas Using Scan Line Based Segmentation," *arXiv preprint arXiv:1603.00912*, 2016.

[102] P. Wei, J. E. Ball, and D. T. Anderson, "Multi-sensor conflict measurement and information fusion," *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*. International Society for Optics and Photonics, 2016, vol. 9842, p. 98420F.

[103] P. Wei, J. E. Ball, and D. T. Anderson, "Fusion of an Ensemble of Augmented Image Detectors for Robust Object Detection," *Sensors*, vol. 18(3), no. 894, 2018, pp. 1–21.

[104] P. Wei, J. E. Ball, D. T. Anderson, A. Harsh, and C. Archibald, "Measuring conflict in a multi-source environment as a normal measure," *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2015 IEEE 6th International Workshop on*. IEEE, 2015, pp. 225–228.

[105] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[106] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, 1965, pp. 338–353.

[107] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *European Conference on Computer Vision*. Springer, 2014, pp. 818–833.

[108] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity search: the metric space approach*, vol. 32, Springer Science & Business Media, 2006.

[109] Q. Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. IEEE, 2004, vol. 3, pp. 2301–2306.

[110] G. Zhao, X. Xiao, and J. Yuan, "Fusion of Velodyne and camera data for scene parsing," *Information Fusion (FUSION), 2012 15th International Conference on*. IEEE, 2012, pp. 1172–1179.

[111] M. Zorzi and A. Chiuso, "Sparse plus low rank network identification: A nonparametric approach," *Automatica*, vol. 76, 2017, pp. 355–366.

[112] R. Zwick, E. Carlstein, and D. V. Budescu, "Measures of similarity among fuzzy concepts: A comparative analysis," *International Journal of Approximate Reasoning*, vol. 1, no. 2, 1987, pp. 221–242.