Implementing a Library to Calculate Surrogate Safety Measures

Mohammad Hossein Nazemi

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

November 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Mohammad Hossein Nazemi**
Entitled:	**Implementing a Library to Calculate Surrogate Safety Measures**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

____ Chair
Dr. Weiyi Shang

____ Examiner
Dr. Sudhir Mudur

____ Supervisor
Dr. Yann-Gaël Guéhéneuc

Approved by ____
Lata Narayanan, Chair
Department of Computer Science and Software Engineering

______________ 2021 ____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Implementing a Library to Calculate Surrogate Safety Measures

Mohammad Hossein Nazemi

Context. Road traffic is continually increasing The traveled miles of vehicles increased by almost 10% from 2010 to 2019 in the USA . With increase in traffic, the risk of collisions increases as well. The Bureau of transportation statistics (BTS) of the USA reports that crashes data has increased by 24 percent from 2010 to 2019. BTS reports 2,740,000 injuries and 36,096 fatalities in USA in 2019. One means to reduce the number of crashes is by analysing the safety of roads. The main parameters to evaluate the safety of roads is statistical analysis of historical crash data. However, crashes do not happen frequently and, thus, do not help predict future crashes. Therefore, the literature introduced the concept of surrogate safety analysis, which uses various measurements, other than the number of crashes, to evaluate the safety of roads.

Problem. One such measure is post encroachment times (PETs). PETs are defined as time difference between the departure of a road user from an encroachment zone and entry of another road user in the the same zone. PETs are computed by analysing in real time traffic. PETs can be computed among vehicles but also between vehicles and pedestrians and cyclists. Several approaches already exist to compute PETs but have limitations, in particular accuracy, generality, and the practicality of their solutions. In addition to PET values, speed is another metric that can be used to determine the severity of conflicts. Calculating the average and momentary speeds of motorized road users can be beneficial in case studies.

Solution. This thesis presents a novel algorithm and its implementation, in the form of a library, to calculate PETs in real time. This library can use any type of detection and perception technologies, including lidars, optic cameras and radars as sources of inputs, and classical or new deep learning techniques as object detection and classification algorithms. Also, it can detect and calculate PET between any types of road users with any types of movements. The hyper parameters of the library are customizable and can be tuned for different scenarios. Moreover our library calculates average and momentary speed of road users to demonstrate more information for PET conflicts.

Validation. We implemented and deployed our library on the infrastructure of our industrial partner, BlueCityTechnology. BlueCityTechnology uses Lidar technologies to monitor road intersections in several cities world-wide. We set up an experiment to manually validate data at multiple intersections. Using the result of this manual validation as ground truth, we validated our library and reported 86.29% F1-score for our PET detection module .

Conclusion. Our library improves greatly the state-of-the-art and is currently used in real-world applications. Municipalities have been using PET reports of BlueCity solution to conduct before and after case studies to detect and solve possible problems at intersections.

Acknowledgments

Someday, I hope we will reach a time where no one loses a loved one in a car accident. That is the reason that I focused on this subject matter and many people helped me along the way which I would like to mention below.

First, I would like to express my deepest appreciation to my supervisor, Professor Yann-Gaël Guéhéneuc. His insight into my research helped me many times. I cannot speak highly enough of him as a researcher and a kind and caring person. He gave me the needed enthusiasm to do my research and guided me along the way.

I am also grateful to my loving wife, Sama. If it had not been for her, I would not have completed this work. She inspired me with her passion and taught me patience. During my most disappointing moments, she stood by my side and helped me find my goals again.

I'm deeply indebted to my parents. They provided me with the opportunity to study away from home. They accepted my absence in their most needed times. It is their care, love, and endless support that has brought me to where I am today.

Furthermore, I wish to express my gratitude to Dr. Asad Lesani and Mr. Ian Boyd, co-founders of BlueCity Technology. My gratitude goes to the trust they placed in me. My research would not have been possible without them.

My dear friends Saber and Mehran deserve a special mention as well. They both are like brothers to me. I can recall the hundreds of times I talked with them, and their support helped me to persevere. I am fortunate to have two good friends such as them.

Contents

List of Figures

List of Tables

Acronyms

AADT Annual Average Daily Traffic. 10

BTS Bureau of Transportation Statistics. 5

CAV Connected and Automated Vehicles. 10

GDOT Georgia Tech Department of Transportation. 11

PET Post Encroachment Time. 1, 6

SIU Standard Input Unit. 14

SSA Surrogate Safety Analysis. 1, 5

SSM Surrogate Safety Measures. 1, 6

TTC Time to Collision. 8

UAV Utilized Unmanned Aerial Vehicle. 9

Chapter 1

Introduction

Context. One of the important topics in transportation engineering is measuring the safety of roads and intersections. The primary way to measure road traffic safety historically was the number of crashes that happened, over a chosen period of time, at an intersection (Chin & Quek, 1997). However, crashes do not happen frequently, which may downplay dangerous intersections. Also, it is really hard to find the reasons behind crashes by only looking at crash data. It is even harder to distinguish between random occurrence of crashes and statistically dangerous location that causes crashes (Hydén, 1987; Svensson & Hyden, 2006). Moreover, crash data do not show the behaviour of road users that led to the crashes (Laureshyn, Svensson, & Hyden, 2010).

An alternative way to measure safety is using surrogate safety measures (SSM) and surrogate safety analysis (SSA). SSA evaluates and finds conflicts between road users based on their trajectories and movements. There are many SSM, but one of the most reliable and most useful SSM is Post Encroachment Time (PET) (Ismail, Sayed, Saunier, & Lim, 2009). The PET represents the time difference between the departure of a road user from an encroachment zone and the entry of another road user into the same zone. Gettman and Head (2003) showed the effectiveness of PET to measure a severity of a conflict. In the rest of this thesis, we contribute to the detection and calculation of PET.

Problem. With the improvement of technology and sensors, traffic engineers can calculate SSMs using computers and algorithms and, thus, perform automated conflict analysis, which can be useful for before and after safety studies (Sayed, Ismail, Zaki, & Autey, 2012). There are many case studies and pieces of research that evaluate the safety of pedestrians, cyclists, and vehicles on roads using PET and other SSM based on different types of sensors, e.g., (Wu, Abdel-Aty, Zheng, Cai, & Zhang, 2020),(Fu, Miranda-Moreno, & Saunier, 2016),(Alhajyaseen, Asano, & Nakamura, 2012). Optic cameras, Radar, Lidar, are some of the most used sources of inputs. In almost all case studies, researchers used tailored, hand-crafted detection algorithms and developed a PET calculation module from scratch for their particular input. As further explained in Section 2, current work on computing PET has the following limitations:

- **Problem 1:** Coupled to a specific input type and a specific object detection algorithm.
- **Problem 2:** Only suitable for batch/offline process of data.
- **Problem 3:** Use center of objects to obtain their tracks and calculate PET.
- **Problem 4:** Calculate PETs between only specific types of road users.
- **Problem 5:** Need heavy calibration and setup procedure.
- **Problem 6:** Do not consider/handle noises as inputs.

Many companies in the transportation industry have created different sensors and modules for detecting road users. Having a decoupled/generalized PET calculator library can help calculate the PET from their input detection. Road safety requires a PET library that calculates PET regardless of the input types.

Practical application of SSA requires real-time processing. Inputs of the sensors and detection algorithm must be processed continuously and actively 24/7. Previous work did not consider analysing real-time data, and they all have a data-collection step in which the input is brought to the processing unit in advance. Therefore, having a library to calculate PET faster than frame rates of the input and the detection algorithm is necessary.

Also, input noises can effect the overall result significantly. Specially for real-time Edge-based practical applications with limited computational power, noises are common as the output of detection and must be handled.

Mohanty, Panda, and Dey (2021) showed the usage of the PET analysis between cars on location beside of intersection like median opening area on roads. However, most of previous research and algorithms only calculated PET for specific road types and at specific locations, especially intersections. In addition, most of the previous works use center based tracks to detect conflicts, however, this approach can reduce the precision for larger road users, such as buses and trucks.

Calculating PET for any road users and locations requires a flexible module that would not depend on the characteristics of the environment in which a sensor is installed. The PET calculator module should be dynamic and fast enough to be used in any location with none or just small calibration. It should detect conflicts accurately and calculate the time difference precisely.

Solution. To address all the mentioned problems, in this thesis, we propose a generalized and decoupled PET library that can work on any road location and detect any conflicts regardless of the road user type. This library is lightweight and can be used for real-time calculation. It can compute PETs for objects up to 25 frames per second. To increase detection accuracy, we propose a new algorithm to detect and calculate PET based on the bounding boxes of the road users. In addition, we introduce multiple techniques to eliminates noises and decrease the faults of object detection algorithms outputs.

Moreover, to increase the relevancy of our PET and enrich our library, we also introduce a separated momentary and average speed module. Many studies use vehicle speed alongside of PET to evaluate the safety of different road users. As an example, Alhajyaseen et al. (2012) uses speed and PET to evaluate the safety of the pedestrians at intersections with left-turning vehicle manoeuvres. Also, Fu et al. (2016) showed the effectiveness of other safety parameters like speed alongside with PET for evaluation of different treatments for a road user safety.

validation. We perform multiple experiments to evaluate the performance of both the PET and Speed library based on out industrial partner's sensors and object detection and classification algorithm, BlueCityTechnology. In order to conduct our experiments, we annotate 45 minutes of Data for one of the BlueCity sensors. We use three annotators to annotate data separately. We investigate and resolve any inconsistency in the results of our annotators. Using the created ground truth by our annotators, our experiments test the performance of our PET library with different settings. Also, we show the effectiveness of our noise cancellation algorithm and impact on the overall results.

Section 1.1 provides some background information and definitions. Section 2 summarizes the related work. Section 3 describes our approach while Section 4 its validation, including the building of ground truth. Section 5 discusses our approach, its validation, their strengths and weaknesses. Finally, Section 6 concludes with future work.

1.1 Background

The Bureau of transportation statistics (BTS) of the United States Department of Transportation (2019) reported over 6 million motor vehicles crashes in 2019. It also reports over 2 million injuries and over 36 thousand death for the same year.

Transport Canada (2019) also reports over 1 thousand and six hundred casualties caused by collisions in 2019. It categorized the cause of crashes into eight categories: distraction, speed / driving too fast, impaired / under the influence, fatigue, other human factor, environmental factor, vehicle factor, and no contributing factors. Transportation engineers use traffic safety analysis techniques to categorize crashes, analyze the reasons behind the crashes, discover the dangerous locations, and solve possible environmental reasons, like bad traffic signal timing. There are two main methods to analyze the safety of roads:

(1) Analysing the historical crash data

(2) Surrogate safety analysis (SSA)

Researchers use statistical methods to fit a safety function on the historical crash data. These statistical models are used to estimate the safety of traffic facilities and roads (Gettman & Head, 2003). However, analyses based on historical crash data method suffer (Ismail, Sayed, & Saunier, 2010) from the following issues:

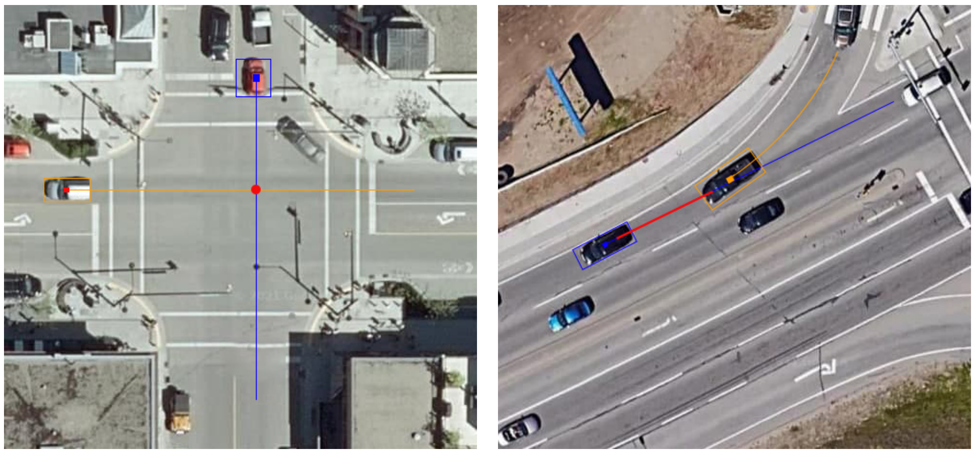
(1) It is hard to attribute crashes to causes. Usually, crash reports do not include causes and are more focused on the responsibilities.

(2) Crashes do not frequently happen, especially for locations with lower traffic volumes.

(3) The data are usually biased towards more damaging crashes

The mentioned shortcomings force transportation engineers to adopt new approaches to analyze safety.

In a Surrogate Safety Analysis (SSA), instead of observing the crash data to analyze the causes, engineers observe some measures proven to be correlated with collisions. These measures can provide more information compared to the reported crash data.



(a) conflict point (b) conflict line

Figure 1.1: A conflict point and a conflict line displayed on GoogleMap

Several traffic safety measures have been introduced in the literature, which were reviewed by Gettman and Head (2003). He has comprehensively reviewed some of the most important parameters in his work.. Most of them are based on **traffic conflicts**.

There are many interpretations and definitions of **conflicts** in different countries. Nonetheless, *Proceedings : first workshop on traffic conflicts, Oslo 77* (1977) defines a conflict as a situation where two or more vehicles are at risk of collision if their movements remain unchanged. However, if conflict participants do not collide because of maneuver of both or either of them, then it is called a **conflict event**. This definition can be expanded to conflicts between vehicles and non-motorized road users as well.

Conflict events can happen in a specific point and time, like two cars that cross a single point in intersection perpendicularly which is called a **conflict point**. Figure 1.1(a) shows a conflict point between two vehicles. Moreover, conflicts can happen in a range of times and locations, like a right turn from a minor street conflicting with the straight movement of a major street as shown in Figure 1.1(b). This type of conflict is called a **conflict line** (Gettman & Head, 2003). For both conflict points and lines we can evaluate different surrogate safety measures (SSMs).

Post Encroachment time (PET) is one of the SSMs that are calculated on conflict points. PET is the time difference between the departure of the first road user and the arrival of the second one at a conflict point. PET is one of the measures used to evaluate the severity of a conflict (Gettman

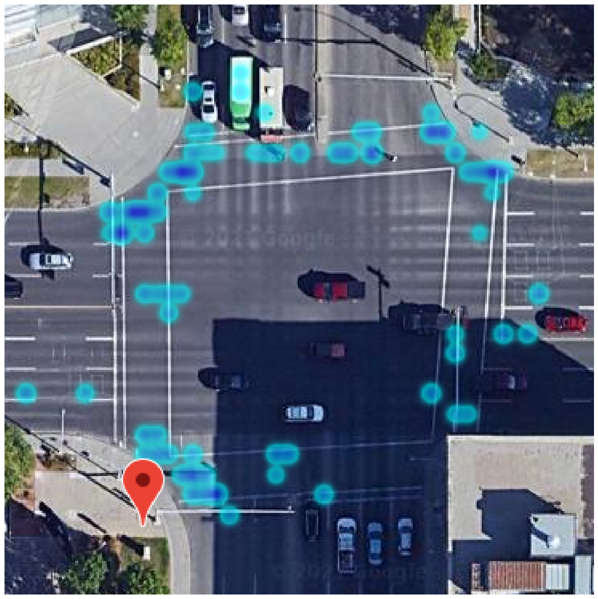


Figure 1.2: Demonstrating the PETs less than 2 seconds between vehicles and pedestrians, displayed on the google map, BlueCity panel, Edmonton, Canada

& Head, 2003) using four threshold values to indicate the severity of a possible conflict:

(1) PET $\leq$ 2s: is called critical threshold and indicates a high risk of collision.

(2) 2s $<$ PET $\leq$ 3s: is a dangerous threshold.

(3) 3s $<$ PET $\leq$ 5s: is low-risk conflicts.

(4) 5s $<$ PET $\leq$ 10s: is called interactions and shows a low possibility of collisions.

Each PET threshold value indicates the severity of the conflicts. By analyzing the locations and the number of conflicts that have happened within each threshold, safety issues can be discovered and resolved. Figure 1.2 displays a heat map showing the number of conflicts with PET less than 2 seconds between pedestrians and vehicles in a day in one of the BlueCityTechnology sensors. The figure shows that the number of PETs on the northwest side of the intersection is higher than in other places, which may indicate a potential problem at the intersection.

Chapter 2

Related Work

The concept of PET has been used widely to measure the safety of different types of road users, including pedestrians, cyclists, and motorized objects. Rather than proposing new methods for calculating PET, most previous work examined its uses to assess traffic safety. Indeed, there is still a lack of libraries with separate, decoupled, accurate PET calculators.

Gettman and Head (2003), using simulation, created a surrogate assessment model to identify conflicts based on their vehicles trajectory data. They showed the possibility of using surrogate safety measures, like PET, to measure the severity of a conflict.

Zangenehpour et al. (Zangenehpour, Strauss, Miranda-Moreno, & Saunier, 2015) developed a solution for calculating PETs based on the tracks of road users. The presented SSA library, however, was designed only for cyclists and vehicles conflicts. It did not consider bounding box of road users and detects conflicts only based on collision of tracks. They used the tracking algorithm presented by Saunier and Sayed (2006), and used the center point of objects to calculate tracks of the objects.

Shared spaces between cyclists and pedestrians were considered by Beitel, Stipancic, Manaugh, and Miranda-Moreno (2018) who aimed to measure the safety using PET by utilizing recorded videos. However, their proposed solution is not real time.

The focus of Paul St-Aubin (2013) was on time to collision (TTC) measure more than PET. Having deployed an automated SSA module, they studied the effectiveness of their proposed system by monitoring the decrease in the number of conflicts in the highway ramps. They found out that rear-end conflicts are more likely to be dangerous on highway ramps, compared to lane changing intersections.

Fu et al. (2016) discussed the challenges of using thermal video data for the safety purposes of pedestrians. They considered interaction between cars and pedestrians in crosswalks using the center of vehicles as the trajectory point. They approach to calculate conflicts between any two objects could be misleading since the conflicts were detected based on the center based tracks of vehicles, particularly for long motorized objects, such as buses and trucks.

The purpose of Chaudhari, Gore, Arkatkar, Joshi, and Pulugurtha (2021) was to discuss the effect of different SSM techniques simultaneously, such as PET and Speed. Although they evaluated the trajectories manually and only for pedestrians and vehicles, they reported that, as the average speed of the vehicles surges, the chance of expecting a conflict involving pedestrians rises substantially.

Zaki, Sayed, Tageldin, and Hussein (2013) identified traffic violations from video sequences. Using the tracks of objects, their approach benefits from matching trajectories for calculating PETs and TTCs. Moreover, they mentioned two noteworthy challenges that future studies face: scalability and the need for more computational resources, and considering different types of errors/noises caused by different weather conditions.

Chen, Zeng, Yu, and Wang (2017) utilized Unmanned Aerial Vehicle (UAV) videos to study conflicts between pedestrians and vehicles. Considering both TTC and PET for safety evaluation, they demonstrated a high exposure of pedestrians to traffic conflicts both inside and outside crosswalks. They observed that the right-turn manoeuvre of vehicles around the corners could be considered as one of the most hazardous behaviours of motorized objects.

Alhajyaseen et al. (2012) used the tracks of objects from recorded video clips to validate their model considering left-turning manoeuvres to assess the safety of pedestrians. As a result of using both PET and vehicle speed at the crosswalk as validation parameters, they found a higher number of critical PETs at compact intersections compared to wider intersections. The slower vehicle speed, on the other hand, decreased the severity of collisions at smaller intersections.

Mohanty et al. (2021) discussed the applicability of PET in different locations besides intersections. They studied the median opening of roads to predict the severity of road crashes. Even though it is obvious that crashes happen rarely and provide insufficient information, this study showed that various locations could benefit from SSA. Consequently, there was a vital need for a comprehensive modules which takes into account different locations and accepts any type of input data.

Peesapati, Hunter, and Rodgers (2018) investigated the power of PET model and traffic volume characteristic data for crash prediction. They showed that estimating car crashes, even in the absence of crash data, was a possibility. They also found that PET can independently perform an intersection diagnosis by categorizing them in various safety categories. However, it was better to use a model combining exposure (Annual Average Daily Traffic, AADT, or conflicting volume) and PET to predict crashes.

Similarly, Sayed et al. (2012) considered the value of before-and-after safety evaluations, particularly when an automated analysis of traffic conflicts is taken into account. Having used an automated traffic safety tool for measuring traffic conflicts provided by the input video, the ramp to the right in an intersection was closed and replaced by a mandatory right-turn lane accompanied by a no-right-turn-on-red sign. Their results showed that this correction leads to a significant decrease in the number of collisions, particularly for rear-end and merging conflicts.

Wang, Xie, Huang, and Liu (2021) discussed effectiveness of the SSM, like PET, on Connected and Automated Vehicles (CAV) safety modeling and evaluation.

All of the above-mentioned studies investigated using PET or similar SSMs to either detect conflicts or predict possible collisions. This particularly demonstrates the importance of more in-depth studies for proposing different approaches of measuring PET. Hereafter, we discuss some articles in which this problem was discussed comprehensively.

Wu et al. (2020) and Zheng (2019) used road users bounding boxes to calculate PET. However, their approach was not efficient because they checked every pixel of each frame. Also, their calculation were not intended for real-time/continuous applications. Their input source were video clips from unmanned aerial vehicles. Despite this, the most important problem was that noise was not taken into consideration. As a result, their solution might not be as accurate as it is in their case studies for general applications.

Ali Kassim and Hassan (2014) calculated PET specifically between cyclists and motor vehicle on bicycle path. They used the front of the rear of the objects and the calculating point of the trajectory. For validation, they used two different observers and if the values were different, they asked a third observer (main researcher) for the third time evaluation. Their approach did not include real time applications and they did not consider potential noises in detection algorithms. Their solution suffers from the same shortcomings that are mentioned for Wu et al. (2020) and Zheng (2019).

Maddox (2016) used Georgia Tech Department of Transportation toolkit (GDOT) to detect and classifies road users. They used the output of GDOT and identified conflicts on video clips. Unlike most of the works in literature, GDOT filters out false positive detections. However, their approach used center-based tracks of objects to detect conflicts rather than considering bounding boxes. Moreover, their solution only calculated PETs between vehicle and pedestrians. Although this work can be used for practical applications due to its lightweight calibration, it did not address real-time/continuous needs for most practical solutions on the market, and was more useful for manual case studies.

In the following we propose an approach to overcome the reviewed shortcomings in the literature.

Chapter 3

Approach

The approach presented in this study covers two different SSMs: PET and Speed. In what follows, each of which will be described in detail. In the table 3.1 we summarize all of the parameters we used in this approach.

3.1 PET

This section aims to cover different parts of the PET calculation module. The proposed approach is to satisfy the following criterion, which each solves the problems mentioned in the introduction:

- **Decoupled from input - addresses problem 1:** Regardless of the sensor device, the proposed

Table 3.1: Notation

Variable name	Used module	Description
t	PET detection	Maximum conflict margin
Q	PET detection	A queue for received SIUs
$Conflict\ Matrix$	PET detection	The main variable to find conflicts and calculate PETs
b	PET detection	The number of SIUs that are kept in Conflict Matrix
b_1	PET detection	The current SIU when calculating PET
r_i	PET detection	The angle of the object corresponding to id i
t_i	PET detection	The timestamp of id i
n_1	Noise removal	Minimum number of SIUs for an object to be valid
n_2	Noise removal	Delayed SIU buffer length
d	Noise removal	Termination threshold
$pixel - meter$	Speed module	converter unit of each sensor

module must be able to process the input.

- **Real-time/Continuous calculation - addresses problem 2:** Not only does the proposed module have to support offline batch processing of data, but practical applications, such as BlueCity, also require continuous and real-time processing of PET.
- **Generalized PET detection algorithm - addresses problem 4:** We demand our module to detect both motorized-motorized conflicts and non-motorized-motorized conflicts.
- **Light calibration - addresses problem 5:** Since a variety of locations need to be monitored, the presented module should be easily calibrated for different types of locations. These types include, but are not limited to, intersections, highways, median openings, and crosswalks.
- **Accuracy - addresses problem 3 and problem 6:** Clearly, the most vital factor of this system is to act as accurately as possible.

To achieve these goals, we designed and developed a novel algorithm that includes three sub-modules:

(1) A serializer sub module to decouple the solution from the input,

(2) A lightweight sub module based on a novel algorithm to detect PET conflicts and calculate PET values based on the bounding boxes of road users,

(3) Noise removal sub modules utilize multiple techniques and algorithms to reduce noise and increase the detection accuracy.

The input of the module can be the output of any detection and classification algorithms. Our developed PET module can use any detection and classification algorithms. In this thesis, we use the BlueCity Lidar solution.

The process of the PET Detection module starts with changing the input to the desired format. There are two noise cancellation sub modules, one before the main PET module and another after it. These sub modules are optional and can be disabled based on the users' preference.

The figure 3.1 illustrates a high-level pipeline of our PET module. We omitted the details of each sub-module to make the figure more readable. In addition, the gray areas of the figures shown

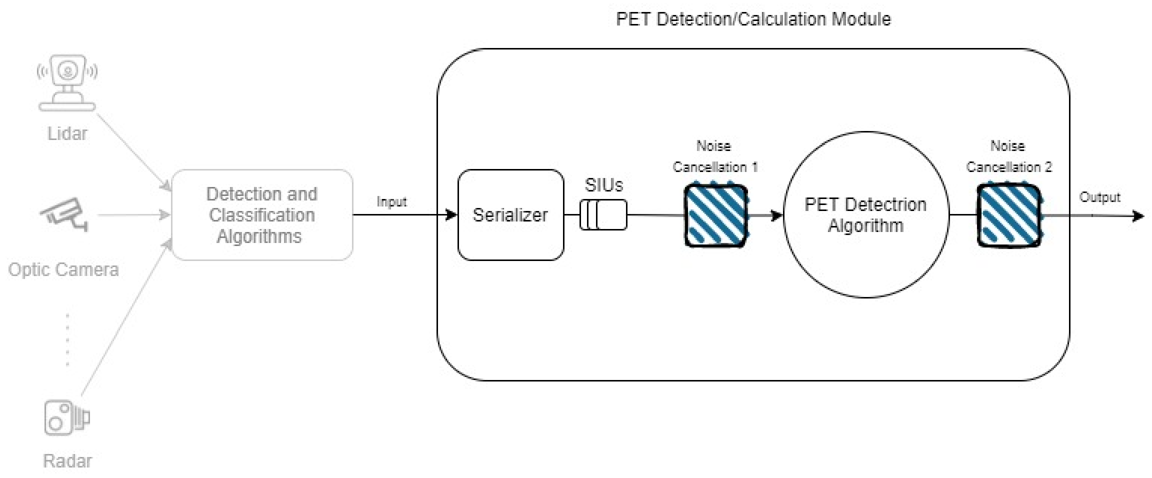


Figure 3.1: Basic PET module pipeline

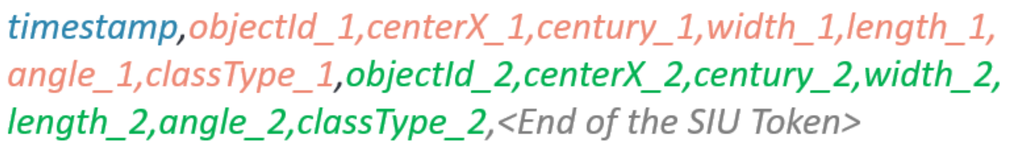

timestamp,objectId_1,centerX_1,century_1,width_1,length_1,
angle_1,classType_1,objectId_2,centerX_2,century_2,width_2,
length_2,angle_2,classType_2,<End of the SIU Token>

Figure 3.2: A *SIU* schema with two objects in it

in this report are beyond the scope of our research. Hereafter, these sub modules are described thoroughly.

3.1.1 Serializer

In practical applications, companies use various sensor devices as the input for their solution. The output of these sensors will be processed by a detection and classification algorithm. By standardizing the output of the algorithms using a helper library, our proposed solution achieves a proper decoupling. Serializer sub module works as a helper library for the detection algorithm and creates an input unit for our algorithm called *standard Input Unit* or *SIU*.

A *SIU* is a simple text that starts with a timestamp followed by the details of multiple detected road users at a frame, called *objects*. Commas separate the *objects*' attributes. A *SIU* ends with a predefined token. The features of each object could be as follows: ID, center position, width and height of the corresponding bounding box, class type, accuracy of the detection algorithm if applicable, angle of the bounding box on the 2D plane of image, speed and acceleration of the object if applicable.

Figure 3.2 displays an instance of a SIU that consists of two objects. If frame rate of the sensor is 24 frames per second, the Serializer sends 24 *SIUs* per second to the detection algorithm. A SIU can be transmitted to the PET module over the Internet or passed to the PET module as an input parameter of the object detection algorithm. This sub module is the first piece of our complete module.

3.1.2 PET Detection Algorithm

Algorithm 1 Detect PET

```
while $True$ do
    $SIU$ ←fetchNewSIU()
    for all object in SIU do
        $calculatePET(object)$
        StoreAttributes(object in)
    end for
end while
```

The input of our PET detection algorithm is a *SIU*. Generally, PET has two consecutive steps: conflict detection, and PET value calculation. The goal of the module discussed here is to cover both. Moreover, the main data structure that we use to detect conflicts between different objects is called *conflictMatrix*. We describe *conflictMatrix* more in depth while we go through the algorithm.

The conflicts are detected by several comparisons between the current SIU's objects and the previous SIUs' objects, up to t seconds. The variable t is the maximum PET threshold. This value depends on the different use cases and can be set based on the maximum PET value the user desires the algorithm to detect. The value of t can be set in a configuration file of the module. By default and based on the literature, the maximum threshold is 10s and PETs higher than 10s are not considered as dangerous interactions.

Our general goal here is to use the bounding boxes of the road users to detect conflicts. To achieve this goal, we store the attributes of each road users, like position (X and Y positions in the frame of the sensor), angle (rotation of the road user in the frame), and type (vehicle, pedestrian, and etc.) at each timestamp. By storing these values, we can compare each road user at timestamp t_1 with the stored values to find potential conflicts. We use a two-dimensional matrix, namely *conflictMatrix*. Variable *conflictMatrix* is a 2d map that stores the occupied pixels of all road users of the last f SIUs. Variable f can be calculated based on the frame rate of the input sensor and variable t:

$$f = t \times fps \tag{1}$$

Algorithm 1 shows the high-level algorithm of the module. This module processes each object in a received *SIU* one by one to find potential conflicts and store the attributes of the objects in the *conflictMatrix*.

The process of conflict detection and PET calculation is broken down into five steps: Fetching a SIU, Calculating AreaPixels, Checking for candidate conflicts, Applying filters and submitting the object in the **conflictMatrix**. Figure 3.3 shows these steps. We discuss each of them in detail in the following.

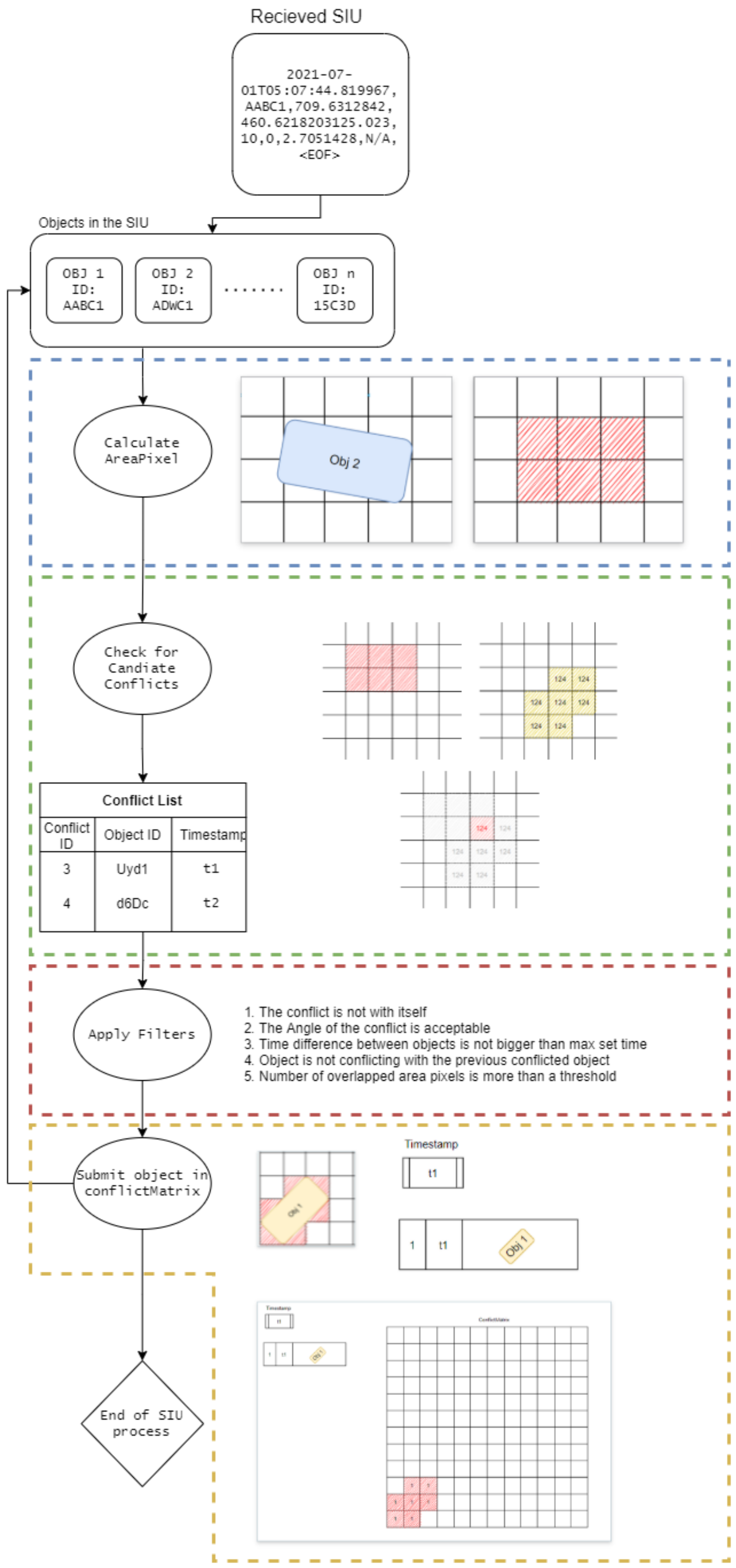
Fetching a SIU

To fetch a new *SIU*, a background process receives the *SIUs* from the serializer continuously and queue them in a fixed-sized queue Q. By calling the *fetchNewSIU()* function, the oldest SIU existing in Q is dequeued, and it will be ready for the next steps.

There are two benefits for having a fixed-sized Q and a separate process for receiving *SIUs*. Firstly, it will avoid memory leakage substantially. Secondly, in a situation where the processing time exceeds the time code[1] and Q reaches its maximum limit, it will prevent the system from halting. The size of the Q can be set based on the available memory of the machine that is running the library.

For example, even if we receive 72 fps from the serializer, our algorithm downgrades the processing SIU rate by skipping some SIUs in Q and module overcomes this particular situation by processing one SIU in each 3 received SIUs.

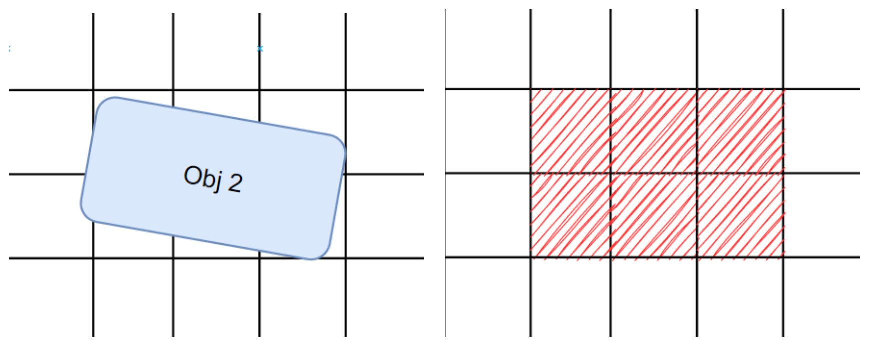
[1]The interval between any pair of frames, e.g., for a 24 fps input, the time code is $\frac{1}{24}$ seconds.



Conflict List		
Conflict ID	Object ID	Timestamp
3	Uyd1	t1
4	d6Dc	t2

1. The conflict is not with itself
2. The Angle of the conflict is acceptable
3. Time difference between objects is not bigger than max set time
4. Object is not conflicting with the previous conflicted object
5. Number of overlapped area pixels is more than a threshold

Figure 3.3: Conflict detection / PET calculation module

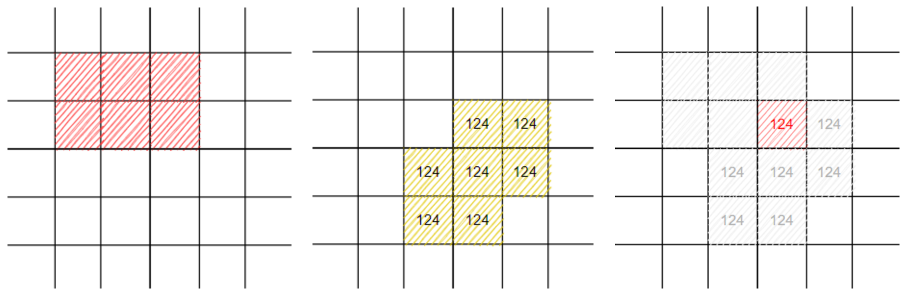


(a) Obj 2 position displayed on **ConflictMatrix** (b) AreaPixels corresponding to the position of Obj 2 on **ConflictMatrix**

Figure 3.4: Calculating AreaPiexls of Obj2

Calculating AreaPixel

For each newly received SIU, our algorithm tries to find potential conflicts and calculates the PETs of those conflicts for each object individually. As discussed before, to achieve this functionality, we used the two-dimensional matrix, *conflictMatrix*. The number of columns of *conflictMatrix* is equal to the width of the sensor output image in pixels. Similarly, the number of rows of *conflictMatrix* corresponds with the height of the sensor output image in pixels. The *conflictMatrix* is a one-to-one map to the sensor's output image. Therefore, each object in a *SIU* occupies some pixels/cells of the *conflictMatrix*. Using the object center position, width, height, and rotation of the bounding box, the pixels that lie within the area of the object's bounding box are calculated. We call these pixels the *areaPixels* of the object. Figure 3.4 demonstrates the **areaPixels** that correspond to a road user.



(a) **AreaPixels** of the object that is being processed (b) ID 124 is for an object that was (c) Current object conflicts with the older object 124 in the red cell

Figure 3.5: Calculating AreaPiexls of Obj2

Checking for Candidate Conflicts

The *conflictMatrix* is filled with IDs of different objects in previous timestamps. We explain the details of how *conflictMatrix* is filled in the last step.

Now that we have the *areaPixels* of the object, we may check for possible overlaps between any two objects that share a part of their *areaPixels*. For each object, the ID that lies within its *areaPixels* is considered its *conflict candidates*.

Suppose object x is being processed at SIU b_1, and it conflicts with ID $1, \cdots, n$. Objects corresponding to ID $1, \cdots, n$ are considered as the conflict candidates. We then retrieve object attributes (position, bounding box, angle and type) and timestamp information corresponding to each ID in the range $1, \cdots, n$. Figure 3.5 demonstrate an example of a conflict between an object and an older object with ID 124. To determine the validity of *candidates* we will apply some filters.

Applying Filters

We apply four filters to detect actual conflicts and remove invalid candidates.

(1) *The candidate is the same object:* An object may have conflicts with its own *areaPixels* in the previous timestamps. For example, ID i might correspond with the *areaPixel* of object x in SIU $b_1 - 1$. In this case, object x and id i do not conflict.

(2) *The angle of the objects is valid:* PET should be calculated when two motorized vehicles cross each others' track, not when they follow each other. PET is only considered for conflict points, not conflict lines. To only detect conflict points, we apply the following test. We denote the angle of object i by r_i. We evaluate the absolute difference between r_x and r_i, if object x and id i are involved in a conflict. If $|r_x - r_i|$ exceeds a predefined threshold, it means that those objects are not following each other. Besides, for a conflict between a non-motorized and a motorized road user, the conflict is always valid regardless of the angle.

(3) *The candidate object has not expired:* We denote the timestamp corresponding to id i by t_i. If $|t_x - t_i| > t$, then the object i has arrived at the conflict point more than t seconds ago and we consider object i as an expired object and we do not consider it for a conflict. Recall that

t is the maximum threshold that the algorithm considers for PET calculation.

(4) *The number of overlapping pixels is valid:* The *areaPixels* of the object x and i should overlap more than a specific threshold to be considered a valid conflict. We describe this test in Subsection 3.1.3, when discussing noise removal techniques.

If a candidate passes all tests, it will be considered a valid PET conflict. By subtracting the active SIU's timestamp and candidate's timestamp, we calculate the PET value of the conflict:

$$\forall i \text{ being a valid conflict; PETvalue}_{<x,i>} = t_x - t_i \tag{2}$$

An object x can conflict with multiple objects at once. Algorithm 3 summarizes the above-mentioned process for calculating PET.

Algorithm 2 submitSIU

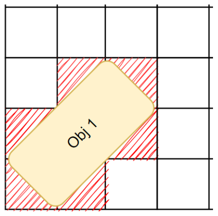
```
Require: SIU,areaPixels
  for all $object$ in $SIU$ do
    $conflictMatrixID$ ←generateID($object$,$SIU$)
    fillConflictMatrix($areaPixels$,$conflictMatrixID$)
  end for
```

Submitting Object in ConflictMatrix

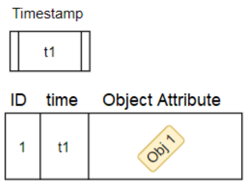
Having found the conflicts of the object x and calculating the PETs, we need to store the attribute of the object x in the *conflictMatrix* and then continue to process the other objects. We generate a unique ID based on the object's attribute and SIU timestamp. This ID will be used to fill the *conflictMatrix*. Figure 3.6 demonstrates an example of submitting process. We also keep track of the unique ID by utilizing a hash map that maps an id to a pair of $< object - timestamp >$. We use this hash map to retrieve the information of the object for potential conflicts in future. Filling *conflictMatrix* with an id might overwrite an existing ID. Algorithm 2 is a pseudo code of the described process.

We now describe the noise removal mechanism which is the last sub module of our developed library for PET detection.

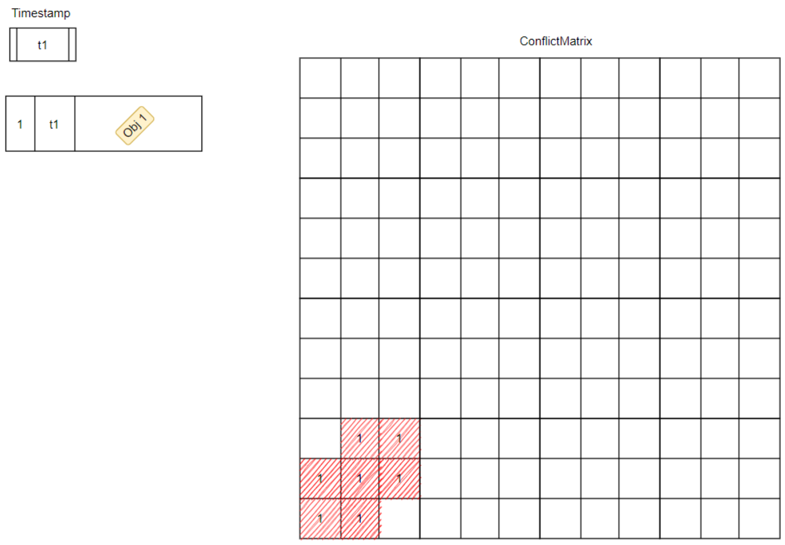
Algorithm 3 shows the process of all steps combined.



(a) **AreaPixels** of the obj1



(b) Creating ID 1 for obj 1 at timestamp t1



(c) Filling the ConflictMatrix with ID 1

Figure 3.6: Submitting Obj1

3.1.3 Noise Removal

Generally, the output of the object classification algorithm could be quite noisy. Even the most elegant object detection and classification algorithms, such as (Bochkovskiy, Wang, & Liao, 2020) and (He, Gkioxari, Dollár, & Girshick, 2017), are subject to noise. To create a generalized PET calculator module, we cannot assume that the input of our algorithm is accurate. Thus, we have to handle noises appropriately.

We observed four behaviours in the output of the Bluecity detection algorithm that could lead to an inaccurate PET detection.

(1) *False positive detection of objects:* Different weather conditions and lighting can mislead the algorithm to detect an object which does not exist,

Algorithm 3 calculatePET

```
Require: object, conflictMatrix
  $areaPixels$ ←getAreaPixel(object)
  $overlappedIDs$ ←getIDs(conflictMatrix,areaPixels)
  for all ID in overlapped IDs do
    if applyTests(ID,object) then
      $PETTime$ ←calculate time difference (ID,object)
    end if
  end for
```

(2) *Inaccurate bounding boxes:* Detected bounding boxes do not fit accurately on the object,

(3) *Inaccurate object class type:* An object's class type might be inconsistent throughout its living cycle.

(4) *False-negative detection of objects:* Sometimes detection algorithms fail to detect an object,

We cannot improve the object detection algorithm's accuracy since we do not have access to the original image being processed. Therefore, except for the last type of noise which is considered beyond the scope of this study, we will delve into the first three.

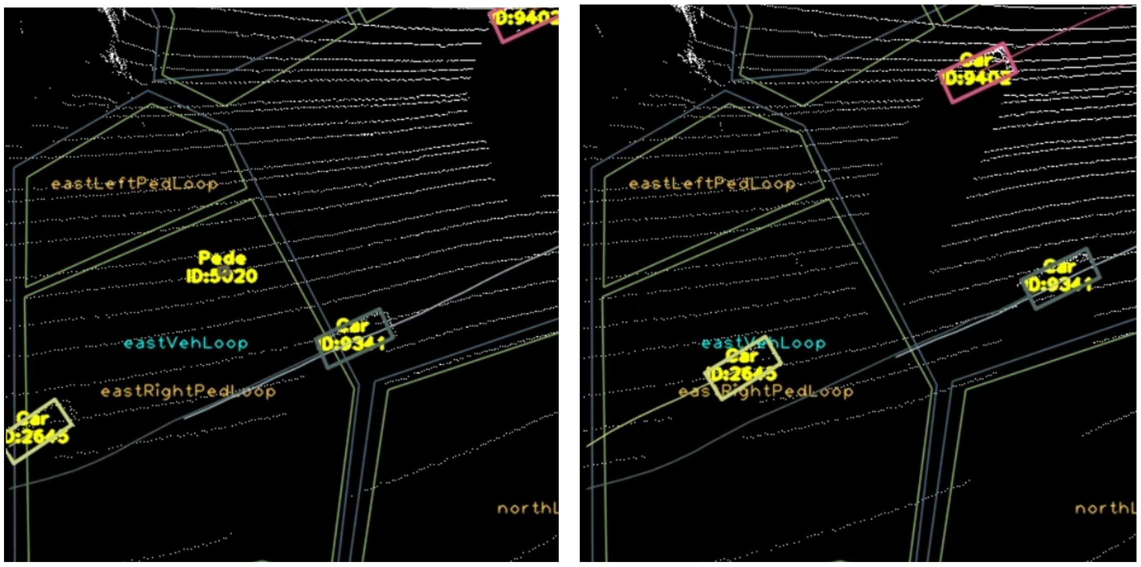
Handling False Positive Detections

Given the sensitivity of the problems in this area and the limited resources available for edge-based solutions[2]

BlueCity company prefers to sacrifice precision to achieve a higher recall, to some extent. Instead of missing an important road user, they identify as many objects as possible in the detection and classification phase. Subsequently, false-positive detections are common and should be considered on the PET detection module.

Figure 3.7 demonstrates an example, in which a pedestrian that was detected in Figure 3.7(a), is disappeared after a few SIUs (3.7(b)). To handle these types of situations, we developed two techniques: Delayed SIUs and Valid Turning Movements. These techniques are discussed hereafter.

[2]Detection and classification could be done on the sensor device itself, rather than being delivered to a server for further processing. This type of solution could be called edge-based.



(a) Detected pedestrian (ID:5020) in the middle of crosswalk (Noise)

(b) A false positive sample (BlueCity Sensor Clip

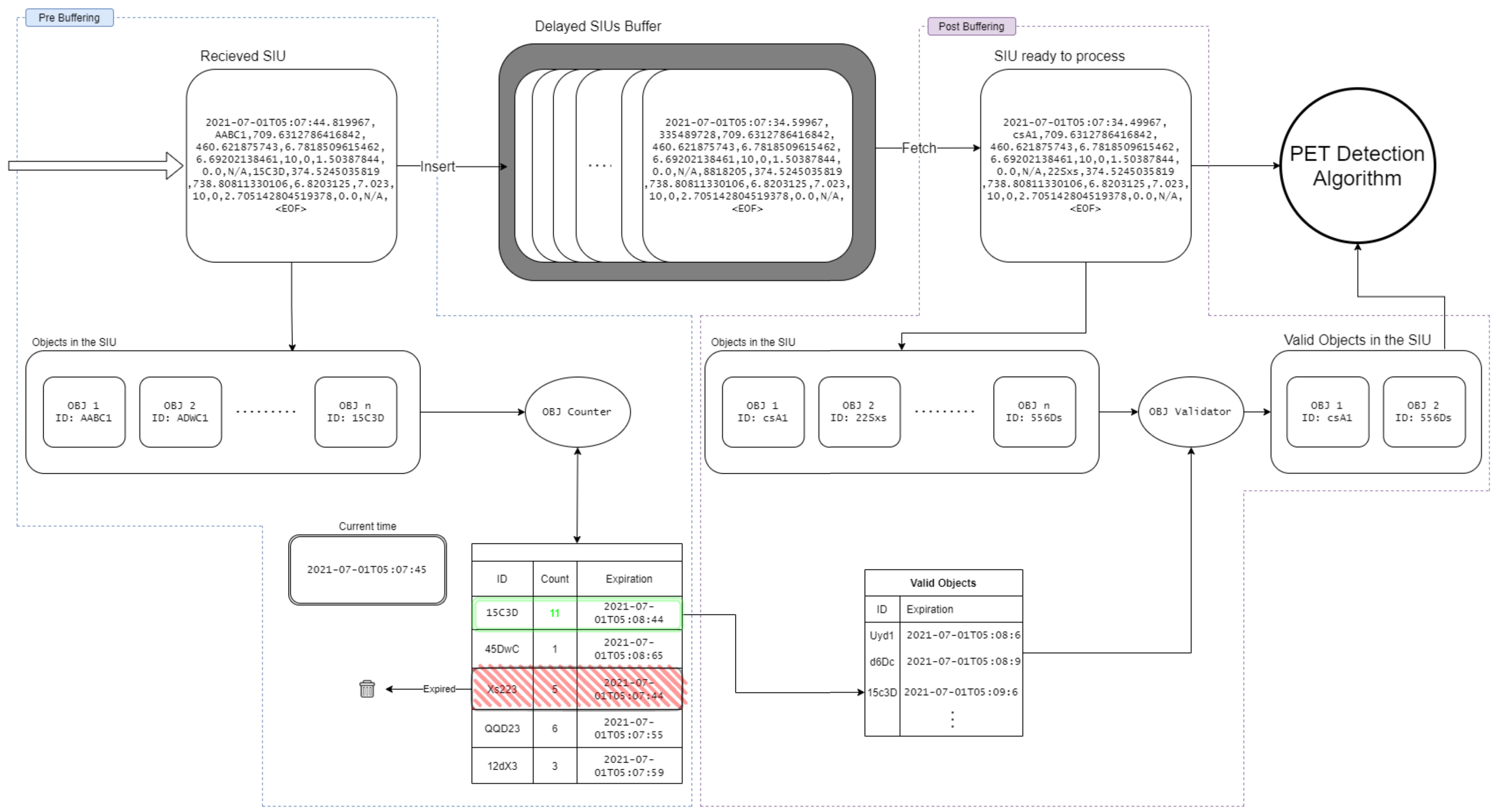
Figure 3.7: Display appearance and disappearance of a noise

Delayed SIUs. The first algorithm that we developed to overcome false-positive noise is “delayed SIUs”. This module sits before the PET Detection module. The idea behind this method is that false-positive instances usually disappear after a few SIUs, while true positive objects live through several SIUs. We call an object a *noise* if it will appear in less than n_1 SIUs in the next n_2 SIUs.

We introduced delayed SIUs as Figure 3.8 illustrates. This technique consists of three stages: pre-buffering, buffering, and post-buffering.

(1) *Pre-buffering:* In the pre-buffer stage, after receiving a SIU from the serializer, the embedded objects are decomposed and transferred to the *Object Counter* module. This module uses a table to count and store the number of times an object appeared in previous SIUs. If *Object Counter* counts an object for at least n_1 times before the object expires, the object ID will be pushed to another table called the Valid Objects table. *Object Counter* removes expired objects from the table to avoid overflowing. We may calculate the expiration date of an object based on the SIU rate and the size of the buffer, n_2:

$$expiration = SIUTime + \frac{n_2}{FPS} + e \tag{3}$$



```
2021-07-01T05:07:44.819967,
AABC1,709.6312786416842,
460.621875743,6.7818509615462,
6.69202138461,10,0,1.50387844,
0.0,N/A,15C3D,374.5245035819
,738.80811330106,6.8203125,7.023,
10,0,2.705142804519378,0.0,N/A,
<EOF>
```

```
2021-07-01T05:07:34.59967,
335489728,709.6312786416842,
460.621875743,6.7818509615462,
6.69202138461,10,0,1.50387844,
0.0,N/A,8818205,374.5245035819
,738.80811330106,6.8203125,7.023,
10,0,2.705142804519378,0.0,N/A,
<EOF>
```

```
2021-07-01T05:07:34.49967,
csA1,709.6312786416842,
460.621875743,6.7818509615462,
6.69202138461,10,0,1.50387844,
0.0,N/A,22Sxs,374.5245035819
,738.80811330106,6.8203125,7.023,
10,0,2.705142804519378,0.0,N/A,
<EOF>
```

ID	Count	Expiration
15C3D	11	2021-07-01T05:08:44
45DwC	1	2021-07-01T05:08:65
Xs223	5	2021-07-01T05:07:44
QQD23	6	2021-07-01T05:07:55
12dX3	3	2021-07-01T05:07:59

Valid Objects

ID	Expiration
Uyd1	2021-07-01T05:08:6
d6Dc	2021-07-01T05:08:9
15c3D	2021-07-01T05:09:6
	⋮

Figure 3.8: PET module with delayed SIUs

In Equation (3), e is a constant variable in seconds to make sure that the fetching and inserting time of the SIU into the buffer does not affect the expiration process. This gives the process a bit of flexibility.

(2) *Buffering:* In the second step, the SIUs are pushed to a first-in-first-out buffer, namely delayed SIUs buffer. The size of the buffer is customizable in the configuration file of the module. Variable n_2 is the buffer size. The first entered SIU is pushed out of the buffer when the buffer size reaches its maximum. The retrieved SIU is processed in a stage called post-buffering stage.

(3) *Post-buffering:* In the last step, a module called the OBJ Validator uses the Valid Objects table to validate the SIU's objects and delete objects that have either expired or are not in the table. We pass the validated objects as well as the timestamp of the SIU as a new SIU to the PET Detection Algorithm.

A pseudo code for the described process is provided in Algorithm 4.

Algorithm 4 DelayedSIU

```
while $True$ do
    $SIU$ ←fetchNewSIU()
    pre-process($SIU$)
    $activeSIU$ ←fetchFromBuffer()
    if $activeSIU$ is not NULL then
        $PETPoints$ ←calculatePET($activeSIU$)
        submitSIU($activeSIU$)
    end if
end while
```

Validating Turning Movements. The second algorithm that is used to reduce false/positive noises is "validating turning movements". Motorized objects at intersections start their movements from one approach and end in another. This behaviour can help us to remove objects that are classified as motorized objects but do not have valid turning movements. For calculating the turning movement of an object, several regions for each approach should be defined manually. Also, we must track the entry point and the end point of each object.

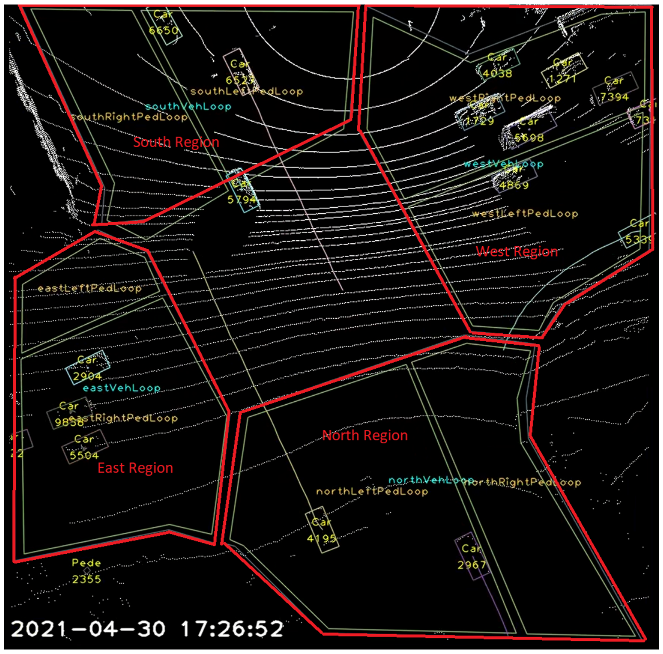


Figure 3.9: Four regions defined on BlueCity lidar image for a sensor in Edmonton, Canada

Figure 3.9 is an example of four defined regions in an intersection. Each region should be wide enough to cover all the movements involved in that approach. This technique is dedicated to motorized classes at intersections only.

Now we will define two new modules, namely Termination Counter and Turning Movement Validator. Figure 3.10 illustrates the process of these modules. Also, Algorithm 5 provides a simple pseudo-code of the algorithm.

(1) *Termination Counter:* The turning movement of an object is calculated by considering the first and the last regions in which a particular object was located. To implement this feature, we must have a module that notifies us of an object exiting the intersection, called Termination Counter. Therefore, this module is introduced after the PET algorithm that handles the status of the objects. If an object is not displayed for d consecutive SIUs, it will be considered as a *terminated* object. Variable d is customizable in the configuration file of the module.

This module goes through the list of valid objects, stores them in the Termination Counts table, and assigns a variable d to each object ID as the initial value. In the next SIU, it iterates over all the IDs stored in the table. If an object does not exist in the current SIU, its d variable is decreased; otherwise, the variable associated with that object is re-initialized to d. Once variable d associated with an object reaches zero, is removed from the table. It is stored on an expiring waiting list, namely *Terminated Objects*. Each ID is removed from this table after a specific amount of time. For each object in the *Terminated Objects* list, we calculate its turning movement by locating its first and last regions.

(2) *Turning Movement Validator:* Turning movement validator can put a hold on the received PET conflicts and wait for the termination of each object. After the termination of each object, the module can validate the conflicts based on each object turning movement and remove conflicts that contain objects with invalid turning movement. This module ignores the turning movements of non-motorized objects.

Algorithm 5 Turning Movement Validator

```
$ValidPETConflictList$
while $True$ do
    $SIU$ ←fetchNewSIU()
    $PETPoints$ ←calculatePET(SIU)
    holdPETConflict($PETPoints$)
    submitSIU(SIU)
    $finishedOBJs$ ←calcFinsihedObjects($SIU$)
    for all $finishedOBJ$ in $finishedOBJs$ do
        $PETConflict$ ←getPETfromHoldList($finishedOBJ$)
        if $PETConflict$ is not $NULL$ then
            SetTurningMovement($PETConflict$,$finishedOBJ$)
            if Both turning movements of $PETConflict$ are satisfied then
                $ValidPETConflictList$ ←PETConflict
            end if
        end if
    end for
end while
```

Handling Inaccuracy in Bounding Boxes

The goal of this module is to limit the PET conflicts that are caused by inaccuracy in bounding boxes. Figure 3.11 shows an inaccurate bounding box assigned to an object. In this figure, the white dense area represents the actual object, while the detected bounding box is shown by a red rectangle that does not fit the object precisely.

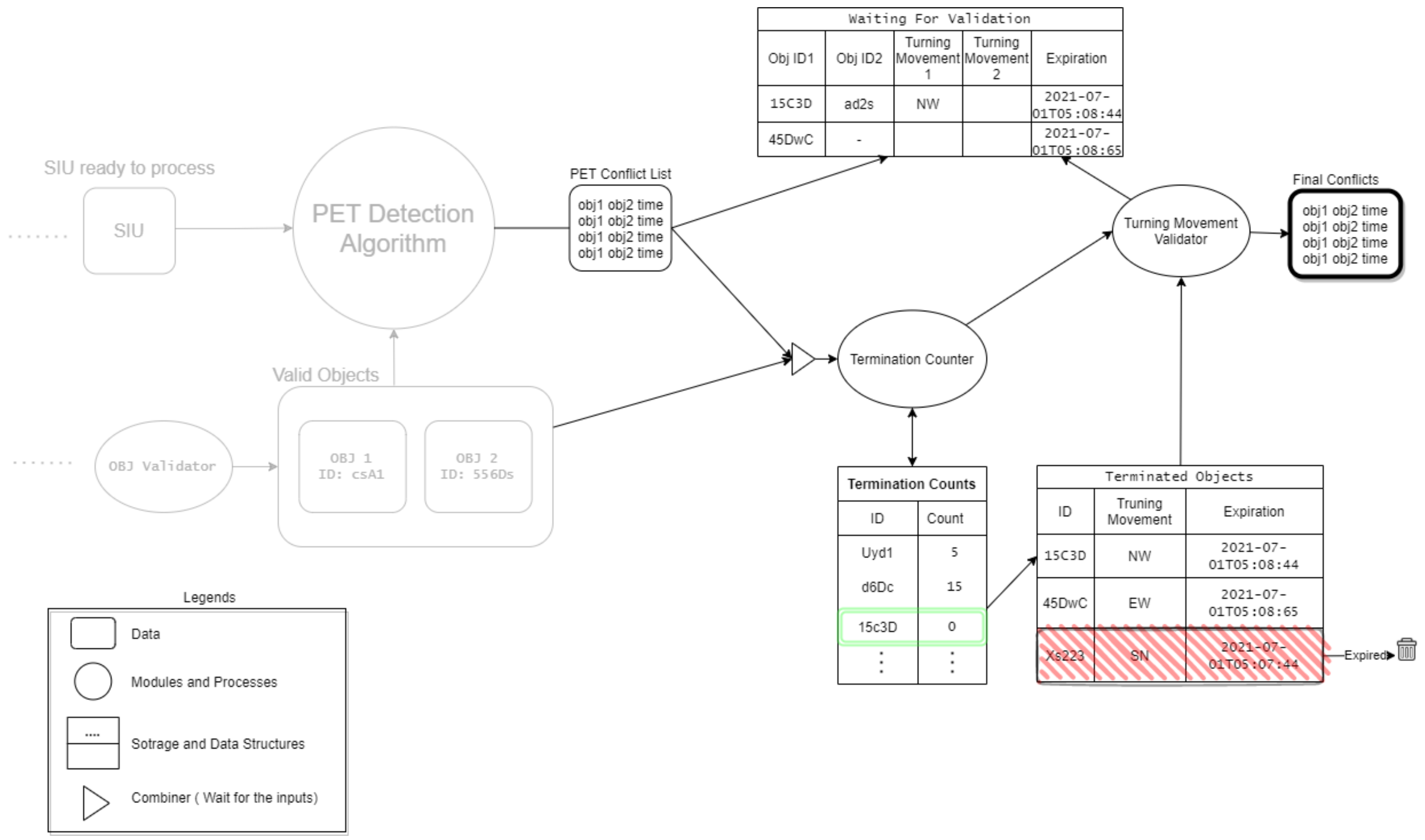


Figure 3.10: PET module with Turning Movement Validator

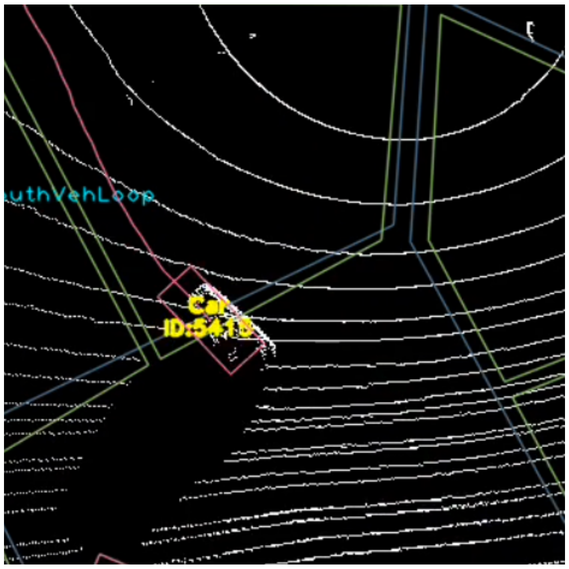


Figure 3.11: A sample of inaccurate bounding box assigned to a car in a frame (BlueCity Sensor)

We cannot increase the accuracy of object detection, because it is out of the scope of this study. However, by defining a minimum percentage of conflict area between two objects, we can eliminate some of the conflicts that have happened because of inaccuracy in the bounding boxes. Applying this technique, however, may add delay to the PET value of conflicts with small overlaps between bounding boxes. So balancing the precision and recall here is a compromise that users need to do depending on their use case.

Handling Inaccuracy in Class Type

The distance of an object to the sensor may affect the detected class type of the object. For example, object x could be classified as a pedestrian at the beginning of its life cycle, and end up in being a motor vehicle. This inconsistency may lead to assigning a wrong type to a conflict involving object x. Several techniques used in this study, including primary PET tests and noise removal techniques, depend heavily on the object type. Therefore, mislabeling could damage the accuracy of our algorithms.

To handle inaccuracy in class types, we introduced the average type technique. When a PET happens, the algorithm caches the PET and its property, but will not submit the result until both objects exit the sensor's frame. A similar procedure to *turning movement validator* that we used object exit/termination module. Meanwhile, we average all the class types labelled to each object from its first appearance in a SIU and when its last appearance. After both objects exit the frames we re-assign the averaged labels to each of them, re-apply related filters to the class types and then submit the PET.

3.2 Speed

In addition to PET calculation, which helps in Surrogate Safety Analysis, another measure that provides more information about the severity of a conflict is the momentary/average speed of the motorized vehicles. For example, by using the momentary speed of the objects involved in a PET conflict, we gain insight into the severity of the conflict. Using the momentary speed and a region representing the intersection that covers it completely, we can also calculate the average speed of a vehicle. Average speed in using momentary speed module. Therefore, we describe the Momentary Speed module first and then take a look at the Average Speed module.

3.2.1 Momentary speed

To calculate the momentary speed of a vehicle, speed calculation module is created. This module takes the current position and the previous position of the object, measures the distance and time difference between two SIUs and calculates the speed of an object using the mentioned parameters.

To calculate the distance of an object in two different SIUs, we first measure the distance in pixel values, then using a converter variable called $pixelMeter$, convert the pixel distance to meter distance. The variable $pixelMeter$ should be calibrated at the initialization stage of the module. The value of this variable depends on the characteristics of the sensor and its installed position and must be set once and only once.

Calculating the velocity of an object using its current position and its previous position can be sensitive and prone to error due to the possibility of inaccuracy in the detection of the object position. To achieve greater tolerance for the mentioned inaccuracy, we utilize the moving average method.

The algorithm not only takes the position of the object in the previous SIU into account, it also considers the s previous SIUs positions and calculates multiple speed values based on each of them. Then, by averaging the calculated values, it reaches to an accurate estimation of the momentary speed of the object. The value of the variable s is customizable in the configuration file of the module. Algorithm 6 demonstrates the described process for $s = 4$.

Algorithm 6 Calculate Speed

```
Require: SIUObject,currentTime
  $currentPos \leftarrow SIUObject.postion()$
  $speeds \leftarrow 0$
  $s \leftarrow 4$
  for $i = 1$, $i{+}{+}$,while $i <= s$ do
    $prevPos \leftarrow getObjPosInPrevSIU(SIUCount - i)$
    $prevTime \leftarrow getObjtimeInPrevSIU(SIUCount - i)$
    $distance \leftarrow (currentPos - prevPos) \times pixelMeter$
    $speed \leftarrow distance/(currentTime - prevTime)$
    $speeds \leftarrow speeds + speed$
  end for
  $movingAverageSpeed \leftarrow speeds/4$
  return $movingAverageSpeed$
```

3.2.2 Average Speed

Average speed is another important parameter that can be used to analyze the safety of an intersection. Averaging the speeds for different turning movements can be used for planning at intersections. For this purpose, we need to define a region for each intersection representing the area of the intersection. For each vehicle inside the region, the module averages the momentary speed of them until they exit the region.

As soon as a vehicle enters the region, algorithms calculate the speed of the object in that SIU based on the previous SIUs position. In order to reduce the effects of inaccuracy in objects detection, we calculate an object speed in a SIU based on its position in the past n SIUs.

Algorithm 7 Speed calcualtion

```
while $True$ do
    $SIU \leftarrow fetchNewSIU()$
    $SIUTime \leftarrow SIU.time()$
    for $SIUObject$ in $SIU$ do
        if $SIUObject.position$ in $speedregion$ then
            $objSpeed \leftarrow calculateSpeed(SIUObject)$
        end if
        if $SIUObject$ has exited $speedregion$ then
            $averagedSpeed \leftarrow calculateAvegrageSpeed()$
        end if
    end for
    submitSIU(SIU)
end while
```

Chapter 4

Validation

In this chapter, we validate our two proposed SSM by comparing our algorithms' results with ground truth. In the following sections, we go through each module's validation steps.

4.1 PET

PET module is implemented on multiple sensors of BlueCity in different cities with various intersection environments. This module calculates PET conflicts in real-time and stores the data on BlueCity databases. To evaluate the performance of our algorithm, we have tested it with multiple settings in three different scenarios.

We gathered 45 minutes of data from one of the BlueCity sensors in Edmonton, Canada. The sensor is located at 104 Ave NW - 109 St NW, Edmonton, Alberta. Each approach of this intersection has 7 lanes and the dimension of the intersection is approximately 20 meters by 20 meters. Figure 4.1 shows a Google maps view of this intersection. In this figure, the red pin is the location of the installed BlueCity sensor.

The intersection could be considered as an intersection with many vehicle and pedestrian traffics, to some extent. We have collected 45 minutes of data for PET detection evaluation on a Tuesday during the rush hour from 9 AM to 9:45 AM (MT).

Due to the nature of the application, data that are coming from BlueCity detection to our PET algorithm has higher recall and lower precision. BlueCity detection algorithm can detect a variety

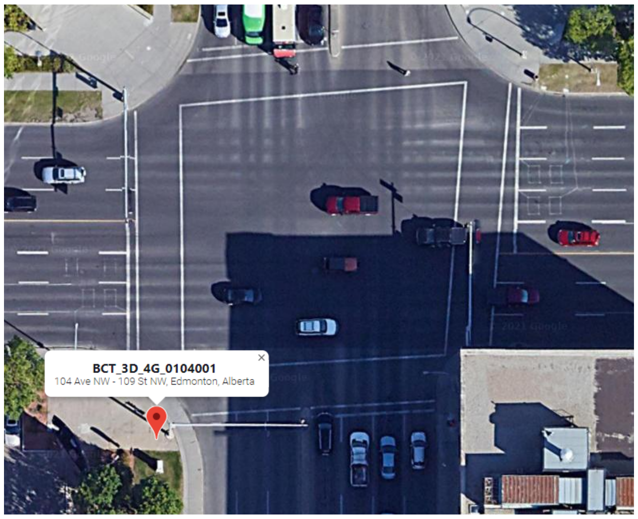


Figure 4.1: The Google maps' view of the considered intersection located at Edmonton.

of objects with different classes, bounding boxes, and speeds. The frame rate of BlueCity sensors is 11 FPS.

In the following, we will discuss the process of our evaluation, tools, the designed experiments, and the results of our tests.

4.1.1 Ground truth

We need to have a ground truth of our data to be able to validate the results of our algorithm. For this purpose, we labelled 45 minutes of data manually, based on the sensor's output video. We created an annotating software that will be discussed in detail in the next section. Using that software, we used 3 individuals' help, including the author of this report, to annotate the data. We recorded a 45-minute video clip of the Edmonton sensor and shared it with our annotators, asking them to use the software to annotate conflicts that occurred in less than 3 seconds. Multiple studies used a 3-second margin as a cut-off value for finding the critical conflicts Gettman, Pu, Sayed, and Shelby (2008). The video clip showed the track of each object for its last 3 seconds, therefore, annotators were able to use this visual hint to annotate conflicts.

If we reach inconsistency among annotators' decisions, we check that specific point for another round, to make sure that it was a valid less than 3 seconds PET conflict. We stored the result as a CSV file and used it as *ground truth* to compare with the result of our algorithm.

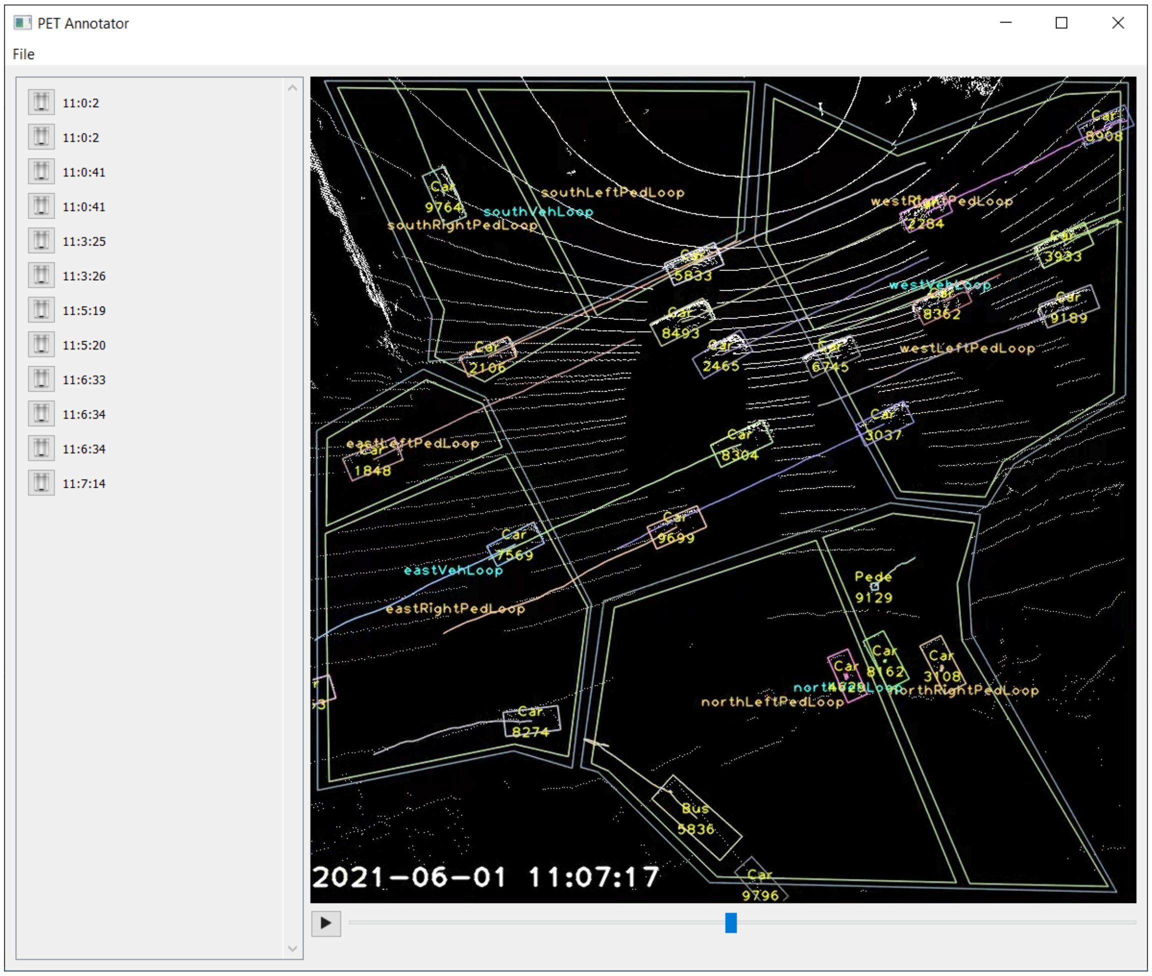


Figure 4.2: PET annotation software)

4.1.2 Annotating tool

Labelling PET conflict is a hard task to do. Annotators either have to be present at the intersection and observe the traffic with bare eyes or look at the video clips and predict object movements and detect conflicts. Because of the definition of PET, a conflict happens between an object and the track of another object. Keeping the track of objects is not easy, particularly when there are multiple objects present at the intersection at the same time. To facilitate this task, we made a simple annotating software and shared it with our annotators. Also, we added a three-second lasting tail to each object, so it would be easier to detect PET conflicts. Figure 4.2 demonstrates a screenshot of the UI of the created software.

We developed this software, keeping in mind the needs of our annotators. The most basic feature of this software is to play video clips. Using this software, we can open different video clips in MPEG format. The software opens the clip in its default resolution. We added the play rate customization feature. Annotators can make the clip slower or faster in different scenarios. The user may conveniently change the frames, skip forward, and rewind the clip.

However, the main feature of this software is the annotating part. By clicking on the page, the software records the clip timing, the clicked position based on the resolution of the clip and saves it in a list. On the left side of the software, the list of the clicked points sorted by timestamp is displayed. Annotators can easily click on the delete button and remove the unintentional errors or inaccurate clicks.

Using the export option on the taskbar of the software, annotators can export their annotate points to a CSV file with a similar structure of ground truth. To create grand truth for our evaluation, we fetched CSV files and compared them together. Since the position and timestamp of annotators clips are not precise, we had to define a tolerance parameter for both clicked position and the clicked timestamp. For clicked position tolerance is +/-40 pixels on both x and y-axis, while for the timestamp the window size is +/-3 seconds. We compared the results of all three annotators considering the mentioned threshold. For any miss-matched points, one of the authors reviewed that specific point again to confirm its validity. It should be noted that we made this software open-source and publicly available so it can be used for future annotating use cases [1].

4.1.3 Reliability

We calculated the Fleiss' kappa inter-rater reliability measure in order to assess the reliability of the annotators data. We used Fleiss' kappa measure because we had more than two annotators. We considered the unique detected PET conflicts as subjects of our measure. Our annotators detected 83 unique conflicts. They labelled 78 similar conflicts. Four PET conflicts were detected only by one annotator and, one conflict was caught by two annotators. Fleiss' kappa measure calculated for this data set is 0.42, which indicates a moderate agreement between our annotators.

However, the measure of joint probability of agreement may better indicate the level of agreement between the annotators because Fleiss' kappa shows paradoxical behaviour in extreme cases

[1] https://github.com/MHNazemi/PETAnnotator

(Feinstein & Cicchetti, 1990), such as ours because we considered only unique conflicts detected by annotators. Moreover, due to the large number of choices that the annotators had (number of conflicts with any PET range), it is extremely unlikely that they would have agreed by chance. Therefore, our annotated data does not suffer from the main shortcoming of the joint probability of agreement, which is the effect of chance on the annotators' results. Thus, we calculated the joint probability of agreement and obtained 0.93 for our ground truth data which is considered as an excellent agreement between our annotators.

4.1.4 Experimental setup

We downloaded the archive data of the sensor detection algorithm for the mentioned date and time and ran the algorithm offline on that data set. We used a computer with 4 GB of Ram and a Core i5 Intel CPU. We have designed three different experiments that will be described hereafter.

Ex1: Impact of noise removal modules on the accuracy

The purpose of Ex1 is to evaluate the accuracy of the PET module. By enabling and disabling noise removal modules, we designed four different modes. The impact of these modes on the accuracy of the PET detection algorithm will be reported. The considered modes are as follows:

(1) Mode 1: without any noise cancellation module,

(2) Mode 2: **delayed SIUs** module enabled,

(3) Mode 3: turning movement filter enabled,

(4) Mode 4: both **delayed SIUs** and turning movement filters enabled.

Note that in all four different modes, the conflict threshold module is enabled, as it does not reduce noises individually and is useful for accurately detecting the PET time.

For evaluating the results, we coded a matching script that looked through the calculated PET points and the ground truth points. It matches the points from each file based on the closeness of timestamps and positions.

Table 4.1: Result of PET detection in one-hour data

	No Filter	Delayed SIUs	Turning Movement	All Filters	Ground truth
Detected PET	731	213	122	68	79
Matched	75	66	66	64	-
False PET	656	147	56	4	-
Missed PET	4	13	13	15	-
Precision	10.25 %	30.98%	54.09%	**94.11%**	-
Recall	**94.93 %**	83.53 %	83.54 %	81.01 %	-
F1-score	18.50%	45.11%	65,66%	**86.29%**	-

Ex2: Processing time

In Ex2 we measured the processing time of our algorithm, considering the four above-mentioned modes. In order to eliminate the fluctuations in our testing machine, we divided our 45 minutes data into three timeslots, every 15 minutes. Eventually, we will report the total processing time spent on each mode.

Ex3: Different settings of delayed SIUs module

In this experiment, we tested different settings of **delayed SIUs** and measured their impact on the total accuracy of the PET module. We used precision, recall and F1-score to show the effect of variable n_1 on the module. We ran our algorithm on the first 15 minutes of our data.

4.1.5 Result

Ex1

The result of Ex1 can be seen in table 4.1. We used 3 criteria of precision, recall and F1-score to compare the results of the cases. Lidar sensors generate sparse cloud points. Accurate real-time detection on the cloud points data can be a heavy task for machines. BlueCity uses edge computing to run its detection algorithm so the amount of resources allocated for this task is limited. The algorithm is biased toward higher recall and lower precision. Therefore, it detects objects in a frame as much as possible but adds false detection that is considered noises in the context of PET detection. False defections can cause false PET conflicts and false insights for cities and traffic engineers. We introduced noise cancellation to overcome this issue.

There were 79 cases of PET conflicts in less than 3 seconds in the 45 minutes of our data set based on our ground truth. As can be seen in table 4.1, in the No filter mode, the algorithm detected 731 PET conflicts less than 3 seconds which is a lot higher than the actual 79 conflicts. This inaccuracy is mainly because of noises that the detection algorithm had generated. This amount of noises affected the precision tremendously, however, the algorithm only missed 4 conflicts. This value for the recall is the highest amount that the algorithm can achieve in different modes. Other modes reduced noise but did not affect the recall value positively.

For the second evaluation, we enabled **delayed SIUs** filter only. This filter was able to reduce noises significantly and improve precision. However, it still did not show a promising result. This filter was suitable for eliminating short-term noises but was not yet able to eliminate long-term ones. Owing to the fact that if we increase the noise threshold of the delay batches, it will be very difficult to distinguish between actual objects and noises.

The third mode was Turnings movement filter enabled. This filter worked better than the other two modes. Both turning movement filter and **delayed SIUs** filter miss-detected 13 conflicts as noises. This happened mainly because of incomplete detection or missing detection on the algorithm side. The main shortcoming of this filter was conflicting between pedestrians and vehicles. We can define valid turning movements for vehicles, however, it is not possible to define valid turning movements for non-motorized objects like pedestrians and bicycles that do not follow a specific path. Therefore, the technique could not filter out the noise that appeared as these types of classes. However, unlike the **delayed SIUs**, the turning movement filter is not dependent on the duration of an object's appearance.

For the last mode, we enabled both of **delayed SIUs** and turning movement filters together to resolve each other shortcomings. By sacrificing a small amount of recall, this technique showed a promising result in terms of precision. This technique filtered both long-lived noises and noises that appeared as pedestrians or bicycles.

To show a better comparison between different techniques we used the F1 score. This showed an excellent score for all filters mode.

Table 4.2: Processing Time of Each Mode

Time	All Filters(s)	Delayed SIUs(s)	Turning Movement(s)	No Filter(s)
15:00 - 15:15 pm	537.61	537.70	635.90	643.00
15:15 - 15:30 pm	485.93	483.63	575.95	567.75
15:30 - 15:45 pm	490.78	490.49	581.68	576.40
Total Seconds	1514.33	1511.82	1793.55	1787.15

Ex2

The purpose of this experiment was to measure the computing time of each mode. Since one of the main goals of this study was to design a real-time PET calculation module, the processing speed of each mode is a critical measure to have. The result of our expriment is shown in table 4.2. Based on the total spent time of each mode, we can see that **delayed SIUs** mode had a faster computational speed in compared to the other modes. The most time-consuming part of our module is finding PET conflicts based on $conflictMatrix$. Since **delayed SIUs** module filters out noises before the PET detection stage, it reduces the number of active objects in $conflictMatrix$ and speeds up the processing time significantly. In the contrast, turning movement does not remove noise before the PET conflict detection stage. Instead, it adds further computations after the calculation of PET conflicts. This extra process requires more time to complete each cycle of the solution and results in slower total processing time in comparison with all other modes. However, by combing **delayed SIUs** and turning movement modules, we could achieve an acceptable result with high-performance accuracy.

Ex3

As we showed in previous studies, **delayed SIUs** plays an important role to increase the performance and the speed of our module. However, the n_1 value can have a significant impact on the performance of this module. This parameter needs to be calibrated based on the specification of each project. We tested different values of n_1 in this study and showed the result in table 4.3 and figure 4.3. As can be seen, by increasing the value of n_1, more objects are considered as noises and less conflicts are detected. To the extent that n_1 equals 100, causes the detection of only one PET

Table 4.3: Comparison of different settings of **delayed SIUs** in all filters included mode

n_1	No. Detected PET	No. Matched PET	Precision	Recall	F1
5	25	19	76%	79.16%	77.55%
10	25	19	76%	79.16%	77.55%
15	22	19	86.36%	79.16%	82.60%
20	22	19	86.36%	79.16%	82.60%
25	20	19	95%	79.16%	**86.36%**
30	20	19	95%	79.16%	**86.36%**
35	20	19	95%	79.16%	**86.36%**
40	20	19	95%	79.16%	**86.36%**
45	20	19	95%	79.16%	**86.36%**
50	20	19	95%	79.16%	**86.36%**
55	17	17	100%	70.83%	82.92%
60	10	10	100%	41.66%	58.82%
65	4	4	100%	16.66%	28.57%
70	4	4	100%	16.66%	28.57%
75	4	4	100%	16.66%	28.57%
80	3	3	100%	12.5%	22.22%
85	2	2	100%	8.33%	15.38%
90	1	1	100%	4.16%	8%
95	1	1	100%	4.16%	8%
100	1	1	100%	4.16%	8%
Ground truth	24	-	-	-	-

conflict. By having a higher n_1 value, the algorithm is less prone to false-positive detection. Therefore, the precision increases. However, many valid conflicts were considered as noises which led to lower recall as the result. By increasing the n_1 to 25, we reached to maximum F1 score that was possible for our algorithm. This score consists of 95% precision and 79.16 % recall. n_1 between 25 to 50 did not change the performance of the algorithm.

4.2 Speed

BlueCity sensors are installed on multiple intersections in different cities. The average speed module is installed for all of the sensors. We have selected a sensor in Edmonton/Alberta Canada for the purpose of data validation in this paper. The sensor is located at 104 Ave NW - 109 St NW, Edmonton, Alberta. Each approach of the intersection has 7 lanes and the intersection is approximately 20 meters in 20 meters. We evaluated around 400 through turning movements of vehicles and compared the calculated speed values with the ground truth.

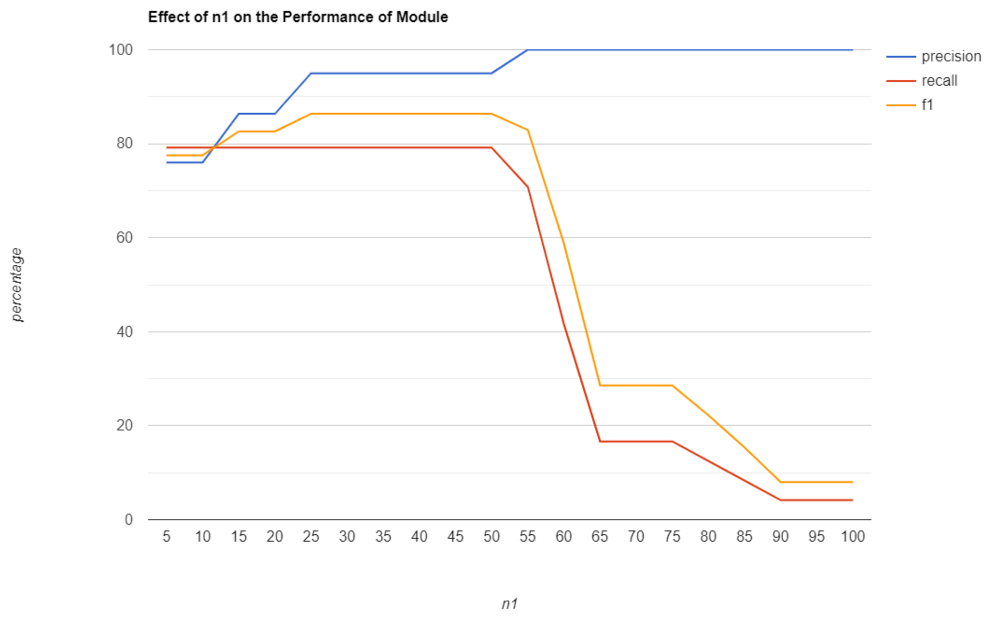


Figure 4.3: n_1 values and its effects on the performance

In the following, we will discuss the process of our evaluation, tools, the designed experiments, and the results of our tests.

4.2.1 Ground truth

For the purpose of our validations for the speed module, we needed to create a data set of through movements speeds as our ground truth. In order to do this, we asked 2 people to help us annotate the data using custom annotation software. We will describe the details of the software in the next subsection. The process of creation of ground truth was as follows:

(1) In order to calculate the average speed of a vehicle, we manually defined a square region that completely covers the intersection. Figure 4.4 illustrates the defined region for our test case. We annotated this region on the 45 minutes video clips shared with our annotators. To be able to calculate the speed of an object in the region, we need to know the entry and exit timestamp of the object, and the distance that it travelled. Therefore, we needed to know the distance

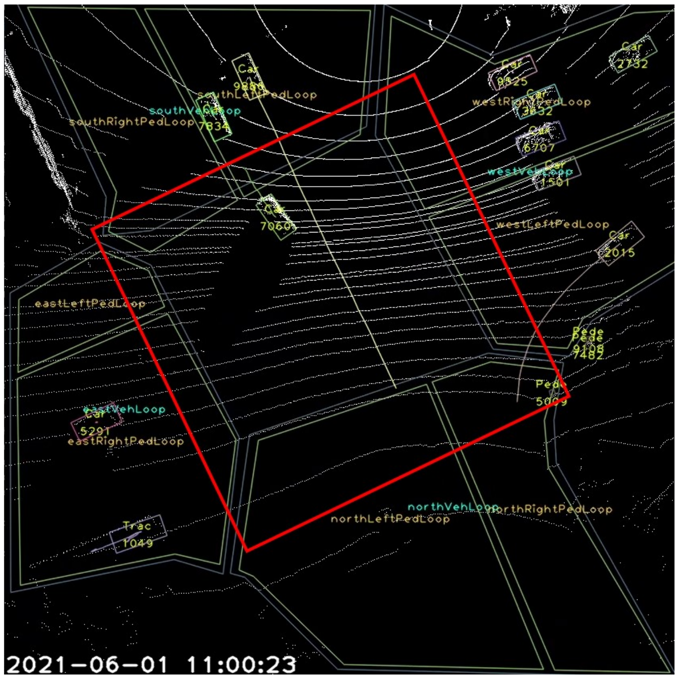


Figure 4.4: Red polygon is a manually defined speed region

between each side of the defined region. Since calculating the distance of turning movement is not an easy task, we only considered through movements that start from one side of the region and finish on the facing side.

(2) For calculating the distance between each facing side of the defined region, we mapped the created region on google maps on the same intersection precisely. Using the measuring distance tool of google maps, we measured the distance between two facing sides of the region in meters. These values were used to calculate the speed of the vehicles.

(3) We asked one of the annotators to select 400 through movements randomly from our 45 minutes data set. To select the movements, he wrote down the vehicle IDs displayed on the top of each vehicle in the clip and timestamp of the vehicle's entry to the regions.

(4) We asked both of the annotators to calculate the average speed of each selected vehicle using our annotation software.

(5) We made the final version of our ground truth by averaging the results of our annotators.

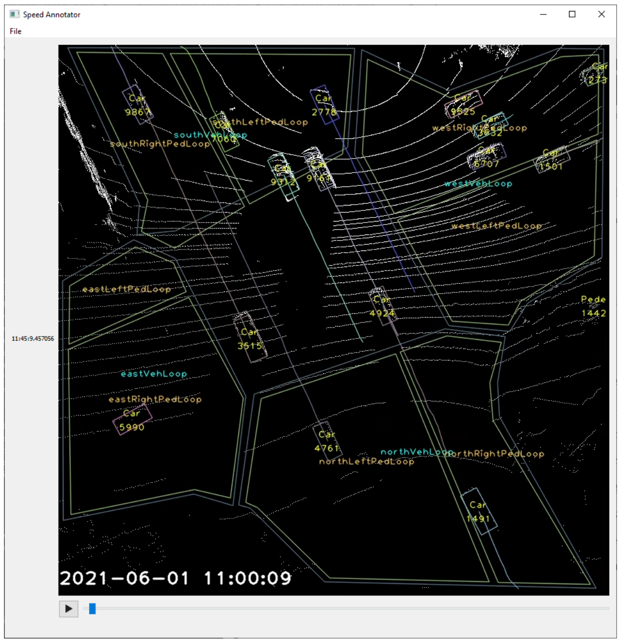


Figure 4.5: Speed annotation software

Based on the described steps, we calculated the ground truth of our average speed study. In the next subsection, we will talk about the annotation software that we created for this purpose.

4.2.2 Annotating tool

Our annotators needed to look at the provided video clips and capture the entry time and exit time of a vehicle to the defined region. For crowded intersections like ours in this study, annotators need to go back and forth into the clips to look at the precise entry or exit time. Also writing down the exact timestamp with millisecond precision is a time-consuming task. For satisfying the mentioned needs, we created custom-made video player/annotation software. This software gives an annotator ability to skip and ... the video clip, change the play rate of the clip, and most importantly, by selecting on the screen, copy the timestamp of the video to the clipboard. Figure 4.5 shows our speed annotation software.

Each annotator reviewed the list of the selected objects. Using the written timestamp for each object, they found the vehicle of interest on the video clip. By clicking on the screen, they captured the arrival and exit time of the vehicle. They pasted those values into an excel sheet containing object ID, the object's turning movement, and the entry and exit times.

Table 4.4: Speed evaluation

Turning Movement	North-South	South-North	East-West	West-East	Total
Count	143	65	95	92	**395**
AVG Speed (km/h)	47.50	43.87	43.13	45.35	**45.35**
AVG Speed GT (km/h)	46.021	43.89	43.37	45.46	**44.90**
Average Abs Speed off (km/h)	1.84	1.14	1.21	1.08	**1.40**
Error %	4.06	2.48	2.36	2.75	**3.09**

4.2.3 Experimental setup

Similar to our PET experiments, we downloaded 45 minutes of our archive data and ran our average speed module. We added the vertices of the considered region to the config file of the module. We used a computer with 4 GB of Ram and a Core i5 Intel CPU for this experiment. We considered only the calculated speeds of the 400 selected vehicles.

4.2.4 Result

Table 4.4 shows the result of our experiment. We removed 5 through movements from our experiment due to their diagonal movements. In other words, they entered the region from either the left-most lane or right-most lane of the approach but exited from right-most the left-most lane respectively. Since we only calculate the distance of straight movements from one side to the facing side of our region, calculating the speed for diagonal movements was not feasible for us.

We separated our data based on the movements of the vehicles. We used average absolute speed off as an indicator of the difference in calculated average speed and ground truth for each movement. As can be seen in the table, the total average speed off was 1.4 KM/h which corresponds to a 3.09% error rate.

Chapter 5

Discussions

This section consists of two parts, threats to the validity of our solution and implementation of our modules on the BlueCity architecture.

5.1 Threats To Validity

During the development of our solution, we aimed to make it as decoupled as possible. Our goal was to design a library that could be used with any type of detection and classification algorithms. Nevertheless, it is important to discuss some of the shortcomings and edge cases related to our solution.

5.1.1 Groundtruth Problem

PET

To obtain ground truth information for our PET experiments, we asked our annotators to annotate videos we shared with them. For each road user in the clips, 3 seconds tail was added. Annotators were tasked with detecting situations where one road user crosses another's path. This task is prone to error since it is possible annotators miss some of the PETs with values near 3 seconds. This missing might cause a false positive result in our solution and decrease the precision of the module.

Speed

As part of our speed experiments, we asked a group of annotators to mark the vehicles' entry and exit time in a predetermined region. This region was mapped to google maps, and using the distance measuring tools of google maps, we fetched the distance between each side of the region. However, the mapping process and also the measuring instrument of Google maps are prone to errors. Consequently, the calculated average speed of the vehicle might not be accurate as well due to a possible error in the distance calculation.

5.1.2 Validation of Our PET Module Accuracy

We were only able to validate the accuracy of our **conflict detection** module. A ground truth consisting of multiple conflict points and their PET values is required to validate the accuracy of the PET calculation module. However, it is difficult to annotate the PET values for each conflict and produce this ground truth. Finding the exact location of a conflict and calculating the time difference between the exiting and entering vehicles is not easy. The best we were able to do was to assign a threshold, find all conflicts below that value, and validate our approach based on that data. We set the threshold to 3s in our experiments, which is considered a critical value in the literature.

5.1.3 Biased Toward BlueCity Solution

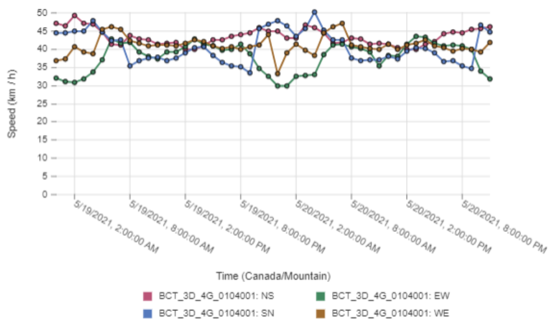
Our experiments were based on the BlueCity solution and sensors. Furthermore, our solution was tested using data collected from BlueCity sensors. Our solution is biased because we could not test it with any other sensors. Additionally, we developed our noise cancellation modules based on the noise we observed on BlueCity's output. We attempted to cover all types of noise; however, the modules and experiments that demonstrated the performance of our modules were all based on the characteristics of BlueCity sensors.

5.1.4 Real-time Problem

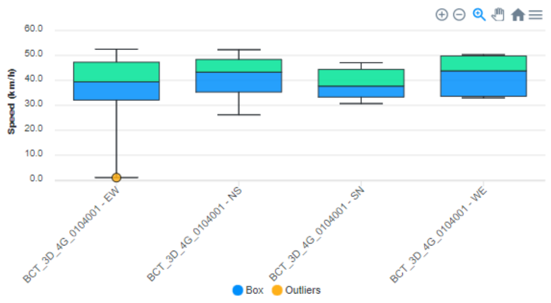
Real-time/continuous operation is one of the key features of our solution. There was no other study in the literature that addressed this practical need. We designed our solution in a way that it can process data frame by frame and calculate PET and speed continuously. Moreover, in order to reduce noises, we developed the **Delayed SIUs** as a part of our noise cancellation techniques. However, this module adds delay to the overall process of data. As an example, if the PET module receives 10 frames per second and the **Delayed SIUs** buffer is 100 frames in size, it adds a 10s delay to the solution. **Delayed SIUs** effects the real-time features of our approach negatively. But it does not have any impact on the continuity of the solution. While this module breaks the real-time feature of our solution, we sacrificed this feature to obtain a more satisfactory result. Moreover, all of the noise cancellation modules are decoupled from the original PET detection and calculation module and can be disabled in the config file easily. Hence, if input noise is minimal, users can disable this module and achieve real-time output.

5.2 BlueCity Implementation

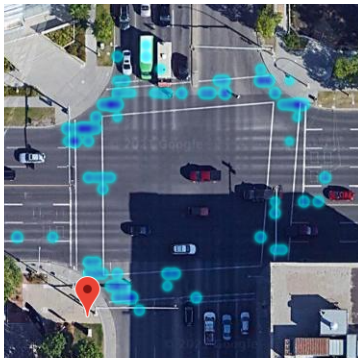
We have implemented and deployed our solution on BlueCity architecture. BlueCity edge-boxes detect and classify road users at intersections using Lidar sensors. We added the serializer module to each edge box. SIUs are generated from the output of the classification and detection algorithm for each Lidar frame. We created a pub/sub-streaming solution to stream the SIUs to AWS EC2 instances that are running our PET and Speed algorithms. Each algorithm is a subscriber to the streaming server, and each edge-box is considered as a publisher. In each instance, our many PET and Speed modules work simultaneously. Both **Delayed SIUs** and **Truning movement validor** are enabled for all of the algorithms. The output of our modules is aggregated and sent to BlueCity Databases periodically. BlueCity panel uses the stored values to demonstrate SSM results on charts. Figures 5.1 show examples of some SSM charts from the BlueCity Panel.



(a) Speed chart- hourly data



(b) Speed chart- Box Plot per approach



(c) PET conflicts and values displayed on Google Maps

Figure 5.1: BlueCity Panel SSM Charts

Chapter 6

Conclusion

In this thesis, we studied post-encroachment times (PETs) and the reasons behind the importance of PET calculation for the safety of road intersections. We found six problems in the literature and designed our modules and sub modules to answer each of them. The problems were as follows:

- **Problem 1:** Coupled to a specific input type and a specific object detection algorithm.
- **Problem 2:** Only suitable for batch/offline process of data.
- **Problem 3:** Use the center of objects to obtain their tracks and calculate PET.
- **Problem 4:** Calculate PETs between only specific types of road users.
- **Problem 5:** Need heavy calibration and setup procedure.
- **Problem 6:** Do not consider/handle noises as inputs.

Also, we talked about the effectiveness of the speed module to show the severity of conflicts.

We introduced a new approach to detect and calculate PET conflicts and PET values. The proposed solution is decoupled from any type of detection algorithm and is suitable for any sensors. In our approach, we used bounding boxes of objects to calculate PETs to achieve more accurate measurement compared to classic approaches that use tracks only. Our technique can process PET conflicts in real-time and compute PETs regardless of objects types.

We also discussed the nature of the input data and the need for noise cancellation techniques. We introduced four techniques to eliminate different types of noises and reduce inaccuracy in PET detection.

In order to calculate the momentary speed of road users, we used moving average techniques. Additionally, by assigning a region to represent an intersection, we calculated the average speed of each road user passing through the intersection.

We deployed our solution on our industrial partner's infrastructure, BlueCity Technology, and used their Lidar to obtain inputs to calculate PETs and speed. For validation, we fetched 45 minutes of data from one of their sensors. We created a ground truth for that 45 minutes of data using custom annotating software and three human annotators. We compared our results with the ground truth in different modes with different noise cancellation techniques enabled. We achieved 94.11% precision and 81.01% recall with all of the noise cancellation techniques and our PET detection technique. We also looked through the different settings of the noise cancellation modules and examined their impact on the result.

In the future, we want to add other surrogate safety measures like Time-to-collision (TTC) to this library. We also need to increase the speed of our PET module for handling frame rates by more than 25 fps. Moreover, we want to test our library on other systems and classification and detection algorithms.

References

Alhajyaseen, W. K., Asano, M., & Nakamura, H. (2012). Estimation of left-turning vehicle maneuvers for the assessment of pedestrian safety at intersections. *IATSS Research*, *36*(1), 66-74. Retrieved from https://www.sciencedirect.com/science/article/pii/S0386111212000167 doi: https://doi.org/10.1016/j.iatssr.2012.03.002

Ali Kassim, K. I., & Hassan, Y. (2014). Automated measuring of cyclist - motor vehicle post encroachment time at signalized intersections. *Nombre de la revista*.

Beitel, D., Stipancic, J., Manaugh, K., & Miranda-Moreno, L. (2018, 12). Assessing safety of shared space using cyclist-pedestrian interactions and automated video conflict analysis. *Transportation Research Part D: Transport and Environment*, *65*, 710-724. doi: 10.1016/j.trd.2018.10.001

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Chaudhari, A., Gore, N., Arkatkar, S., Joshi, G., & Pulugurtha, S. (2021). Exploring pedestrian surrogate safety measures by road geometry at midblock crosswalks: A perspective under mixed traffic conditions. *IATSS Research*, *45*(1), 87-101. Retrieved from https://www.sciencedirect.com/science/article/pii/S0386111220300522 doi: https://doi.org/10.1016/j.iatssr.2020.06.001

Chen, P., Zeng, W., Yu, G., & Wang, Y. (2017). Surrogate safety analysis of pedestrian-vehicle conflict at intersections using unmanned aerial vehicle videos. *Journal of advanced transportation*, *2017*.

Chin, H.-C., & Quek, S.-T. (1997). Measurement of traffic conflicts. *Safety Science*, *26*(3), 169-185. Retrieved from https://www.sciencedirect.com/science/article/pii/S0925753597000416 doi: https://doi.org/10.1016/S0925-7535(97)00041-6

Feinstein, A. R., & Cicchetti, D. V. (1990). High agreement but low kappa: I. the problems of two paradoxes. *Journal of clinical epidemiology*, *43*(6), 543–549.

Fu, T., Miranda-Moreno, L., & Saunier, N. (2016). Pedestrian crosswalk safety at nonsignalized crossings during nighttime: Use of thermal video data and surrogate safety measures. *Transportation Research Record*, *2586*(1), 90-99. Retrieved from https://doi.org/10.3141/2586-10 doi: 10.3141/2586-10

Gettman, D., & Head, L. (2003). Surrogate safety measures from traffic simulation models. *Transportation Research Record*, *1840*(1), 104-115. Retrieved from https://doi.org/10.3141/1840-12 doi: 10.3141/1840-12

Gettman, D., Pu, L., Sayed, T., & Shelby, S. (2008). Surrogate safety assessment model and validation. final report. *National Academy Press, Washington,D.C.*

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 2961–2969).

Hydén, C. (1987). The development of a method for traffic safety evaluation: The swedish traffic conflicts technique..

Ismail, K., Sayed, T., & Saunier, N. (2010). Automated analysis of pedestrian–vehicle conflicts: Context for before-and-after studies. *Transportation research record*, *2198*(1), 52–64.

Ismail, K., Sayed, T., Saunier, N., & Lim, C. (2009). Automated analysis of pedestrian - vehicle conflicts using video data. *Transportation Research Record*, *2140*(1), 44-54.

Laureshyn, A., Svensson, A., & Hyden, C. (2010). Evaluation of traffic safety, based on micro-level behavioural data: Theoretical framework and first implementation. *Accident Analysis & Prevention*, *42*(6), 1637-1646. Retrieved from https://www.sciencedirect.com/science/article/pii/S0001457510001041 doi: https://doi.org/10.1016/j.aap.2010.03.021

Maddox, S. (2016). Using automated video processing to identify pedestrian-vehicle conflicts. *Transportation Research Record: Journal of the Transportation Research Board*.

Mohanty, M., Panda, B., & Dey, P. P. (2021). Quantification of surrogate safety measure to predict severity of road crashes at median openings. *IATSS Research*, *45*(1), 153-159. Retrieved from https://www.sciencedirect.com/science/article/pii/S0386111220300637 doi: https://doi.org/10.1016/j.iatssr.2020.07.003

Paul St-Aubin, N. S., Luis Miranda-Moreno. (2013). An automated surrogate safety analysis at protected highway ramps using cross-sectional and before-after video data. *Transportation Research Part C*.

Peesapati, L. N., Hunter, M. P., & Rodgers, M. O. (2018). Can post encroachment time substitute intersection characteristics in crash prediction models? *Journal of Safety Research*, *66*, 205-211. Retrieved from https://www.sciencedirect.com/science/article/pii/S0022437517302281 doi: https://doi.org/10.1016/j.jsr.2018.05.002

Proceedings : first workshop on traffic conflicts, oslo 77. (1977). Oslo: Norwegian Council for Scientific and Industrial Research.

Saunier, N., & Sayed, T. (2006). A feature-based tracking algorithm for vehicles in intersections. In *The 3rd canadian conference on computer and robot vision (crv'06)* (p. 59-59). doi: 10.1109/CRV.2006.3

Sayed, T., Ismail, K., Zaki, M. H., & Autey, J. (2012). Feasibility of computer vision-based safety evaluations; case study of a signalized right-turn safety treatment. *Transportation Research Record: Journal of the Transportation Research Board*.

Svensson, A., & Hyden, C. (2006). Estimating the severity of safety related behaviour. *Accident Analysis & Prevention*, *38*(2), 379-385.

Transport Canada, R. T. (2019). *Canadian motor vehicle traffic collision statistics: 2019 , contributing factors in fatal collisions.* Retrieved 2021-10-29, from https://tc.canada.ca/en/road-transportation/statistics-data/canadian-motor-vehicle-traffic-collision-statistics-2019

United States Department of Transportation, B. o. T. S. (2019). *Motor vehicle safety data.* Retrieved 2021-10-29, from https://www.bts.gov/content/motor-vehicle-safety-data

Wang, C., Xie, Y., Huang, H., & Liu, P. (2021). A review of surrogate safety measures and their applications in connected and automated vehicles safety modeling. *Accident Analysis & Prevention*, *157*, 106157. Retrieved from https://www.sciencedirect.com/science/article/pii/S0001457521001883 doi: https://doi.org/10.1016/j.aap.2021.106157

Wu, Y., Abdel-Aty, M., Zheng, O., Cai, Q., & Zhang, S. (2020). Automated safety diagnosis based on unmanned aerial vehicle video and deep learning algorithm. *Nombre de la revista*.

Zaki, M. H., Sayed, T., Tageldin, A., & Hussein, M. (2013). Application of computer vision to diagnosis of pedestrian safety issues. *Transportation Research Record*, *2393*(1), 75-84. Retrieved from https://doi.org/10.3141/2393-09 doi: 10.3141/2393-09

Zangenehpour, S., Strauss, J., Miranda-Moreno, L., & Saunier, N. (2015). Are signalized intersections with cycle tracks safer? a case-control study based on automated surrogate safety analysis using video data. *Accident Analysis & Prevention*.

Zheng, O. (2019). Developing a traffic safety diagnostics system for unmanned aerial vehicles usingdeep learning algorithms. *University of Central Florida*.