CONNECTED CARS: GPS/OBD SENSOR FUSION WITH RADIO COMMUNICATION

STEVEN CHEN HAO NYEO

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Thesis Advisor: Dr. Christos A. Papachristou

Department of Electrical, Computer, and Systems Engineering

Case Western Reserve University

January, 2023

CONNECTED CARS: GPS/OBD SENSOR FUSION WITH RADIO COMMUNICATION

Case Western Reserve University
Case School of Graduate Studies

We hereby approve the thesis[1] of

STEVEN CHEN HAO NYEO

for the degree of

Master of Science

Christos A. Papachristou

Committee Chair, Advisor — 12/12/2022
Department of Electrical, Computer, and Systems Engineering

Daniel G. Saab

Committee Member — 12/12/2022
Department of Electrical, Computer, and Systems Engineering

Evren Gurkan-Cavusoglu

Committee Member — 12/12/2022
Department of Electrical, Computer, and Systems Engineering

[1] We certify that written approval has been obtained for any proprietary material contained therein.

Dedicated to Progress in
Computer Engineering

Table of Contents

List of Figures

Acknowledgements

First, I would like to thank my advisor, Dr. Christos Papachristou for his guidance on doing research and technical writing. I can never show enough appreciation for his tireless support throughout my academic journey at Case.

I would also like to acknowledge the enormous help from my friends Jason Paximadas, Shawn Anastasio, Dwarakanath Kottha, and Sonia Joy for providing me insights on implementing the car communication channel through radio and Kalman filtering for sensor fusion.

Last but not least, I would like to thank my parents for their consistent emotional support through college and graduate school.

ABSTRACT

Connected Cars: GPS/OBD Sensor Fusion with Radio Communication

STEVEN CHEN HAO NYEO

Connected vehicle technology (CV) leverages wireless communication and enables cars to navigate traffic in real time. In recent years, the self-driving industry has seen significant technological advancements in connected and autonomous vehicles (CAV), a combination of autonomous vehicle technology (AV) and wireless CV technology. Between these two technologies, there has been a great focus on developing AV technologies that heavily rely on the accuracy of sensors such as cameras, LiDARs, and radar modules. While these sensor technologies are in the process of vehicle integration, our research specifically looks into solutions in current CV technologies that can aid navigation and collision avoidance to complement existing AV sensor technology. This thesis proposes a decentralized wireless framework that can enable two or more connected vehicles to transmit and receive GPS/OBD data such as position, velocity, and acceleration to other vehicles in the vicinity.

1 Introduction

Autonomous vehicle technologies (AV) are the state-of-the-art implementations for self-driving cars and vehicle collision avoidance. AV technology depends directly on peripheral sensors such as but not limited to radar, LiDAR, GPS, and camera vision. These technologies are able to pick up surrounding obstacles quickly in real time but are not immune to all kinds of surroundings. A few examples include radar signals experiencing weak reflections on materials such as concrete, and lane departure systems not being able to pick up lane markings due to the lack of light or color contrast during bad weather conditions.

In the meantime, connected vehicle technologies (CV) [14] emerge to be yet another solution for self-driving cars. Vehicle-to-everything (V2X) communication hypothetically guarantees that all objects in the vicinity of a certain vehicle release wireless signals with moving information. The most prominent technology for creating wireless connections between vehicles is the dedicated short-range communication (DSRC) technology on the IEEE 802.11p wireless channel [1]. Current implementations of DSRC technology are, however, not very cost-effective since they require substantial infrastructure as a prerequisite [19].

Our proposed solution for vehicle collision avoidance is inspired by a combination of both CV and AV. Our system model realizes the potential of vehicle communication by combining GPS data and OBD data along with radio channels as a means of data transmission. Our solution will provide relative object positions that can aid the collision avoidance system should the AV sensors fail to respond to potential collisions.

Our solution also provides a prediction algorithm for triggering alerts when our system model detects the potential of collision by finding the upper and lower bounds of the car position in the next instant from the car's acceleration. Since cars cannot physically accelerate faster than their maximum acceleration and break sooner than their maximum deceleration, the future car position in the next certain unit amount of time is bounded between positions derived from the maximum acceleration when flooring the gas pedal and maximum deceleration when fully applying the brakes.

1.1 Motivation

The motivation for the thesis research is to identify new approaches that can complement existing CV and AV technologies in the self-driving industry, including but not limited to camera vision, LiDAR, radar, and cellular-V2X. However, strong cellular signals are not available in all locations around the world, such as suburban and rural areas. In places with no cellular infrastructure, vehicle data cannot be transmitted through a centralized wireless communication medium. AV sensor technology and imaging processing techniques, on the other hand, can be relatively more reliable than cellular data but are not bulletproof to image processing errors. This thesis builds on top of the decentralized CV networks concept to ensure that the corner cases of existing CV and AV technologies for self-driving cars are addressed.

In terms of implementation costs, high-resolution AV sensors such as LiDARs are extremely costly to integrate with cars. Cellular technology also requires significant starting costs for building large amounts of infrastructure up-front. Moreover, both AV sensors and cellular technology require the modules to be on at all times. AV sensors have to constantly scan for surrounding objects, while cellular networks have to maintain a stable and consistent connection with the cell towers while the vehicle is in operation. Compared to AV sensors and cellular technology, our decentralized framework has the potential to be a much more cost-effective and power-efficient solution.

1.2 Thesis Outline

Chapter 2 of the thesis explains in depth the state-of-the-art implementations of self-driving cars, including AV sensor and wireless CV technologies, and introduces the basics and backgrounds of sensor fusion techniques. Chapter 3 describes the underlying algorithm that calculates the upper and lower bounds of the car position and the different scenarios of how collisions can occur. The simulation results of the chapter 3 algorithm are demonstrated in Chapter 4. Chapter 5 explains the hardware specifications of the system model. Chapter 6 further addresses how the data is collected and processed through the software pipeline, and the test results are analyzed in Chapter 7.

2 Background

2.1 Autonomous Vehicles

In the recent decade, the self-driving automotive industry has been pushing the technological frontiers of creating cars that can effectively navigate through traffic and avoid collisions on the road. Many of these objectives focus on developing more accurate and efficient sensor hardware and software algorithms that aid with a vehicle's ability to detect, identify and learn its surroundings in real time. There are a few common technologies the industry has adopted to achieve this purpose, including but not limited to digital cameras, LiDAR sensors, and automotive radars.

2.1.1 Camera Vision

With digital cameras widely available at low costs nowadays, computer vision technology has been a prominent candidate for deploying autonomous vehicle navigation and collision avoidance. Autonomous vehicle technologies that utilize computer vision systems are capable of processing images in real-time to extract traffic information. Some of the common computer vision algorithms and functionalities used for calculating the desired vehicle trajectory fall into the categories of image filter and enhancement, edge detection for identifying curbs, contour extraction for eliminating unnecessary data points, morphology processing for filling in lossy edge data, and model fitting for classifying identified landscapes with existing geographical templates[29].

However, computer vision heavily relies on the quality of the received imaging. A few scenarios when computer vision has yet to improve include blocked obstacles[29], driving in the dark[10], and inclement weather conditions[21,29]. Camera vision alone is also poor in distance estimation[18] and will require other techniques to identify distinct object clusters and map the objects' distances within the field of view.

2.1.2 LiDAR

LiDARs operate by emitting laser beams and detecting their reflections to capture surrounding object data[18]. These laser beams typically operate at the wavelength of 905 nm, which is within the infrared spectrum. There are two types of LiDARs in the current market: solid-state LiDARs emits a single laser beam to light up the surrounding while scanning LiDARs mechanically rotate the laser beam to scan the environment[24]. LiDARs have the advantage of having high precision and resolution, and the range of LiDARs can extend to hundreds of meters in the distance. In addition, LiDARs can collect data at a very fast rate, giving it the advantage of providing real-time applications in autonomous vehicles[24].

However, the infrared 905 nm wavelength that LiDARs use can damage the human retina under long exposures. Current regulations limit the strength of these infrared waves emitted by the LiDAR modules for human safety defined by the safety standard IEC 60825[?], thus hindering the LiDARs from performing at their maximum capacity[7,18]. The infrared band is also fairly close to the visible light, which makes LiDARs harder to navigate through bad weather compared to other lower-frequency bands[7,18].

Moreover, the LiDAR technology needs to overcome its bulky data size while streaming collected data[24] and significantly higher cost for deployment and integration[7,18,24], and they have yet to meet the reliability standards such as the ISO 26262[2] and IEC 61508[18]. In short, the LiDAR technology is still relatively new compared to other existing technologies and still has some ways to go before the industry sees the scalable deployment of the LiDAR technology on autonomous vehicles.

2.1.3 Radar

Radar modules are able to measure the distance between nearby objects in the field of view by calculating the time difference between the transmitted and received radar signal. Compared to LiDARs which started in the 1970s, radars are a much more mature technology that has existed for over 100 years[7]. In the autonomous driving industry, frequency-modulated continuous wave (FMCW) radars are one of the common radar modules used for adaptive driver-assistance systems (ADAS)[32]. By emitting a continuous frequency ramp known as a "chirp"[3], the FMCW radars can compare the difference between the transmitted and received frequencies from computing range information, as well as decipher velocity information from Doppler phase shifts.

Radar modules cost significantly less to obtain and deploy than LiDAR counterparts and require less computational power during operation[7]. In addition, radars are immune to changes in lighting and environmental conditions compared to pure camera vision or LiDARs[7,32]. In recent years, radars have seen a drastic improvement in resolution, and thus emerging to become a promising candidate for autonomous driving implementations and sensor fusion with camera vision[20].

2.2 Connected Vehicles

Aside from having sensors to aid autonomous driving, research efforts also focus on the communication between vehicles without relying on sensor data. There are two state-of-the-art paradigms for moving vehicles to interact with each other: centralized vs decentralized communication. The centralized communication framework relies on existing networking infrastructure such as network providers and cell towers to manage communication, while the decentralized communication framework has all participating nodes operate on their own on a needed basis.

2.2.1 DSRC

Dedicated short-range communication (DSRC) was introduced to support road safety by sending basic safety messages (BSMs) to the surrounding for other connected cars in the vicinity to detect and process[8]. DSRC utilizes the IEEE 802.11p wireless standard as the basis for network routing and operates on the 5.9 GHz band allocated by the U.S. Federal Communications Commission (FCC)[8,16]. The protocol also heavily depends on vehicles (V2V communication) and roadside units (V2x communication) to frequently exchange data with each other to avoid traffic collisions[16].

The DSRC protocol is a decentralized approach to vehicular detection in that DSRC uses ad-hoc communication for cars to detect and identify other vehicles in the vicinity[4]. The protocol will be able to serve its purpose as long as a nearby vehicle or object has the ability to receive complete and non-garbled data from one to another without relying on the local infrastructure to establish communication. However, DSRC lacks the ability to control and synchronize wireless communication when cars are densely populated in a certain area[4]. Congestion can happen in crowded environments where many nodes attempt to transmit into the same communication channel simultaneously.

2.2.2 3GPP

The 3rd Generation Partnership Project (3GPP) is an organization that standardizes the cellular protocols for mobile telecommunications. 3GPP works on the development of the 4G and 5G technology standards, as well as related standards such as LTE[17]. These protocols rely on the communication channels established between the mobile device and the service provider's cell towers. As opposed to DSRC's decentralized communication framework, cellular communication requires cell towers to act as a centralized node to process data transfer procedures such as packet routing. Therefore, the quality of cellular communication is heavily impacted by the existing infrastructure in the designated area.

In general, LTE latency and packet delivery ratio outperform DSRC especially when there is dense traffic in an area. From the simulations and experiments provided by Bey et al.[6], DSRC shows similar performances compared to LTE when the latency is around 100 ms, but when the latency decreases, the packet delivery success rate decreases dramatically, especially at 10 ms[6].

5G New Radio (NR) is another new technology that could be used for centralized vehicular communications[17]. As cellular communication begins to evolve towards the 5G standard, the implementation of real-time traffic control and collision avoidance will start to benefit from the performance of centralized cellular communication with low latencies. However, the most significant issue with 5G is the backward compatibility of the current infrastructure. 5G technology requires denser deployments of cell towers and sufficient bandwidth to handle the processing of connected autonomous vehicle information.

2.3 Sensor Fusion

Sensor fusion is used to combine various sources of data from sensors with different error rates. Each sensor has its advantages and disadvantages under different operating conditions and these conditions affect the accuracy of the sensor data. To consolidate all sensor information and account for the different measurement errors from each sensor, sensor fusion attempts to produce a result that ideally represents the most accurate sensor value based on the collected data from the different sources. Some common sensor fusion techniques include Gaussian processes, the central limit theorem (CLT), and Kalman filters.

2.3.1 Kalman Filter

Kalman filters[15] are optimal estimation algorithms for scenarios of the system states that cannot be measured directly. The Kalman filtering technique was proposed by Rudolf E. Kálmán in 1960 and has been widely used in comparing theoretical models and measurement data. The simplest Kalman filters are linear Kalman filters, also known as linear quadratic estimation, in which measurement errors are

dynamically adjusted through matrix operations with fixed parameters. There are other variations of Kalman filters such as Extended Kalman filters that tweak these parameters to account for different scenarios with non-linear configurations[15].

Kalman filtering is a common technique for fusing data sets that differ from each other. As it is impossible to keep track of all environmental parameters that may influence the measured output, we can use a Kalman filter approximate the expected output by feeding the input into both the theoretical model and the real model to produce two outputs. The Kalman filter then takes in the two outputs from both models and dynamically adjusts its own parameters according to the sample data until they level off and converge to steady values.

3 System Approach

3.1 Problem Statement

To create a framework independent of local infrastructure and sensor inputs, our solution proposed in this thesis adopts the decentralized wireless framework to create a wireless ad-hoc connection without the need for pre-existing infrastructure. This assumes that as long as two cars have a wireless transceiver, they can establish a wireless connection when they are close to each other. In addition, the data transmitted on this decentralized framework will serve to broadcast location, speed, and acceleration information to nearby vehicles, which helps with calculating the probability of a collision that may occur with the aid of connected car technology.

Our system model, shown in Figure 3.1 requires a hardware environment to test the algorithm's feasibility that calculates the positions of nearby vehicles. The following two chapters will elaborate on both the derivations of the algorithm and the design decisions behind the hardware architecture.

3.2 Prediction Algorithm

In order to predict the short-term trajectory of a car, our solution brings in a car's maximum acceleration and deceleration to predict its position in the upcoming seconds. Assuming that no skidding occurs during the car's movement, the position of the car will be bounded by the area between its maximum and minimum

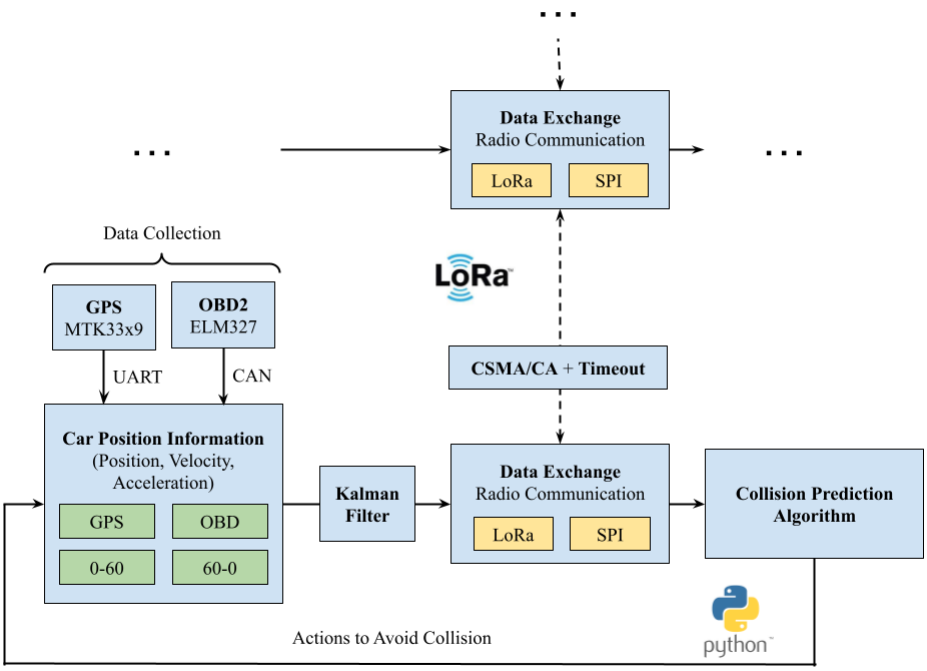


Figure 3.1. System Block Diagram

deceleration. This area guarantees that other cars or surrounding objects can prevent a collision by avoiding the area in the next unit amount of time.

3.2.1 Summary of Cases

This chapter generalizes the collision scenarios into 3 cases: back-end collisions (including head-on collisions), side-way collisions, and corner collisions. These 3 cases cover collisions from all directions between $0°$ and $180°$ and are a 1-dimensional simplification of various yaw rates.

- **Case 1: Back End Collision**

  Figure 3.2 and Figure 3.3 shows the xy-graph for Case 1, where the 2-D surface represents the x and y coordinates of cars A and B. The labels above the position points of A and B represent the time relative to $t = 0$ at the point.

  Case 1 discusses the scenario where the angle between the direction of A and B (θ) = $0°$. Since $\Delta y = y_b - y_a = 0$, we can calculate the distance

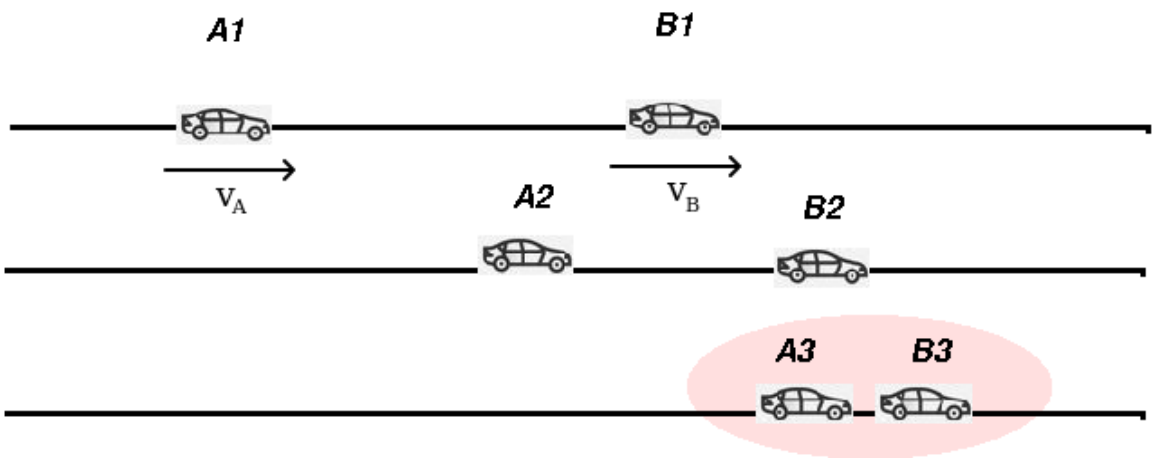


Figure 3.2. Back End Collision

between a and b by:

$$S_{ba} = \Delta x = x_b - x_a$$

Note that collision happens when $x \leq 0$. These collision cases usually happen on highways or straight roads when drivers attempt to pass each other. Case 1 is separated from the other cases since the theoretical trajectories of these collisions are 1-dimensional, which makes it much simpler and straightforward to estimate the time before the collision.

By providing the maximum gas acceleration (a^+) and maximum break deceleration (a^-), we are able to estimate the range of location that the car is going to land in the next certain unit of time, assuming that skidding does not happen.

Figure 3.2 shows the scenario of a back end collision. In this scenario of two cars A and B with specifications of lengths (l_A, l_B), maximum accelerations ($a_A{}^+$, $a_B{}^+$), and decelerations ($a_A{}^-$, $a_B{}^-$), we are given the initial positions (x_A, x_B), velocities (v_A, v_B) and acceleration (a_A, a_B) of each car at time $t = 0$. For a potential collision to happen, the initial condition should satisfy $x_A < x_B$ and $v_A > v_B$. In the case where $a_A = 0$ and $a_B = 0$, the time

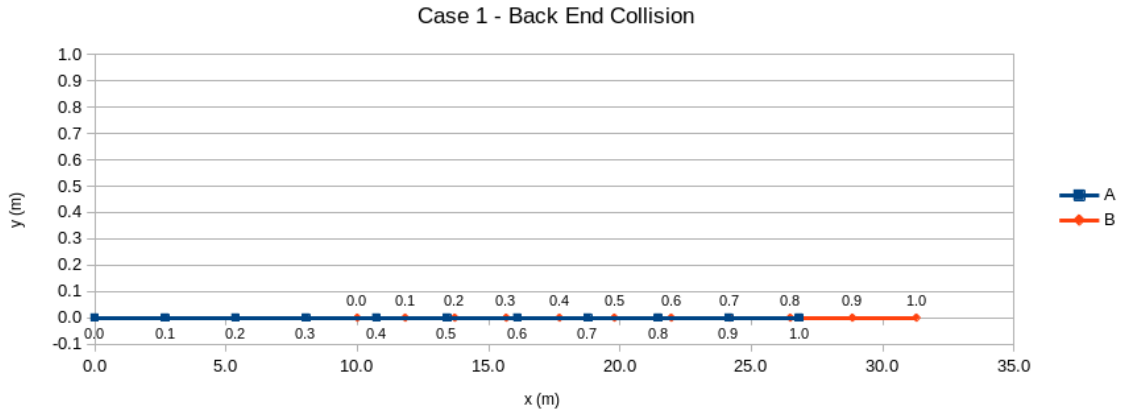


Figure 3.3. Back End Collision Data

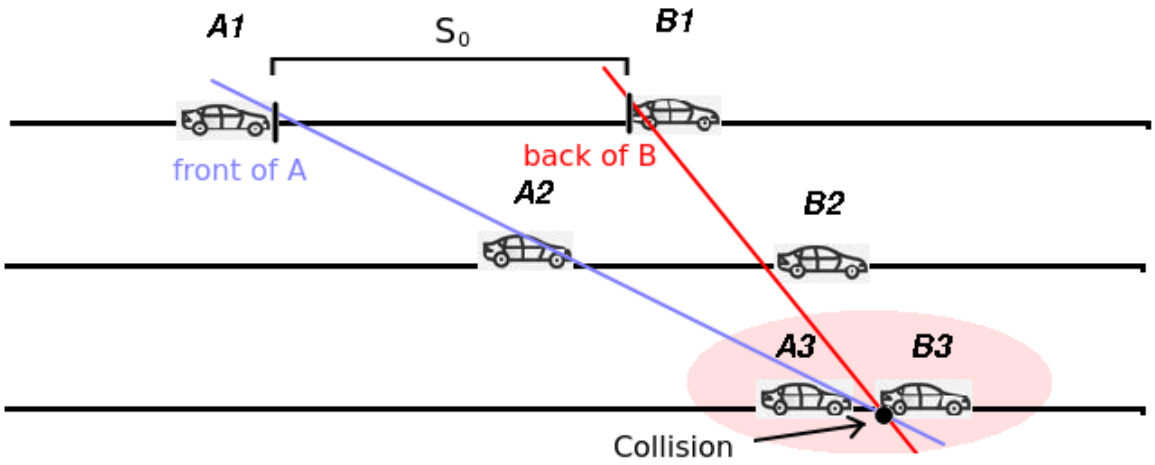


Figure 3.4. Back End Collision when $a_B = 0$

before collision is:

$$t_0 = \frac{S_0}{|v_B - v_A|} = \frac{S_0}{v_A - v_B}$$

where S_0 is the distance from the back of car B to the front of car A denoted by $S_0 = (x_B - x_A) - \frac{l_A + l_B}{2}$, and $|v_B - v_A|$ is the relative speed of car B with respect to A. Figure 3.4 describes the process of collision when both $a_A = 0$ and $a_B = 0$.

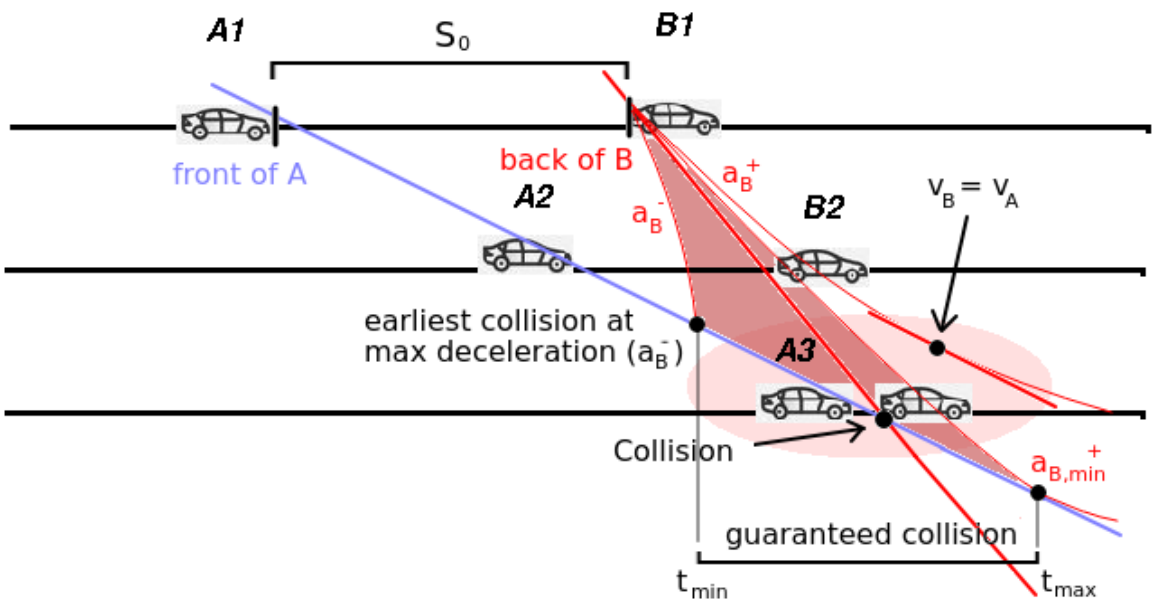


Figure 3.5. Back End Collision when $a_B \neq 0$

In the case where $a_B \neq 0$, we are able to use the kinematic equation ($S = v_0 t + \frac{1}{2}at^2$) to model the change in distance before collision:

$$\begin{aligned} S_0 &= S_B - S_A \\ &= v_B t + \frac{1}{2} a_B t^2 - v_A t \end{aligned}$$

By solving the quadratic equation $(\frac{1}{2}a_B)t^2 + (v_B - v_A)t - S_0 = 0$ for t, we obtain:

$$t = \frac{(v_A - v_B) \pm \sqrt{(v_B - v_A)^2 + 2a_B S_0}}{a_B}, a_B \neq 0$$

By using these roots, we can find the maximum and minimum times before collision (t_{max}, t_{min}):

$$\begin{cases} t_{max} = \frac{(v_A - v_B) \pm \sqrt{(v_B - v_A)^2 + 2S_0 a_{B,min}{}^+}}{a_{B,min}{}^+} \\ t_{min} = \frac{(v_A - v_B) \pm \sqrt{(v_B - v_A)^2 + 2S_0 a_B{}^-}}{a_B{}^-} \end{cases}$$

Figure 3.5 shows the collision scenario when car B is either accelerating or decelerating.

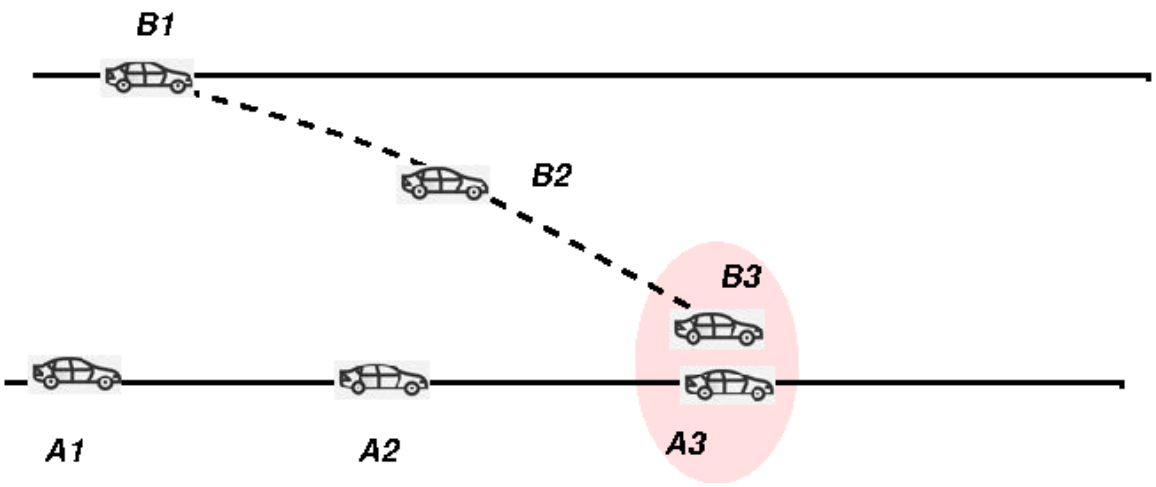


Figure 3.6. Sideway Collision

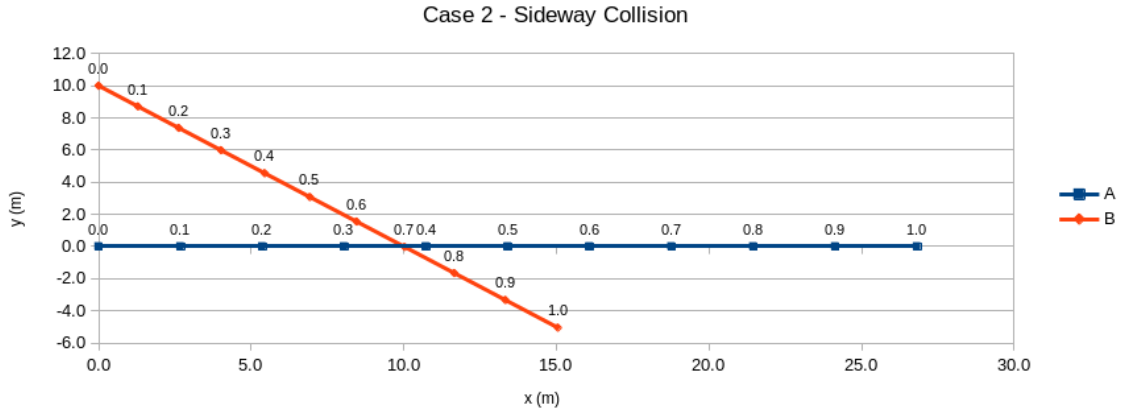


Figure 3.7. Sideway Collision in Action

- **Case 2: Sideway Collision**

  Figure 3.6 and 3.7 depicts Case 2, the scenario when $\theta > 0°$. In this case, $\theta = 45°$. The distance between cars A and B at time t is, therefore:

$$S_{ba} = \sqrt{(\Delta x)^2 + (\Delta y)^2} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

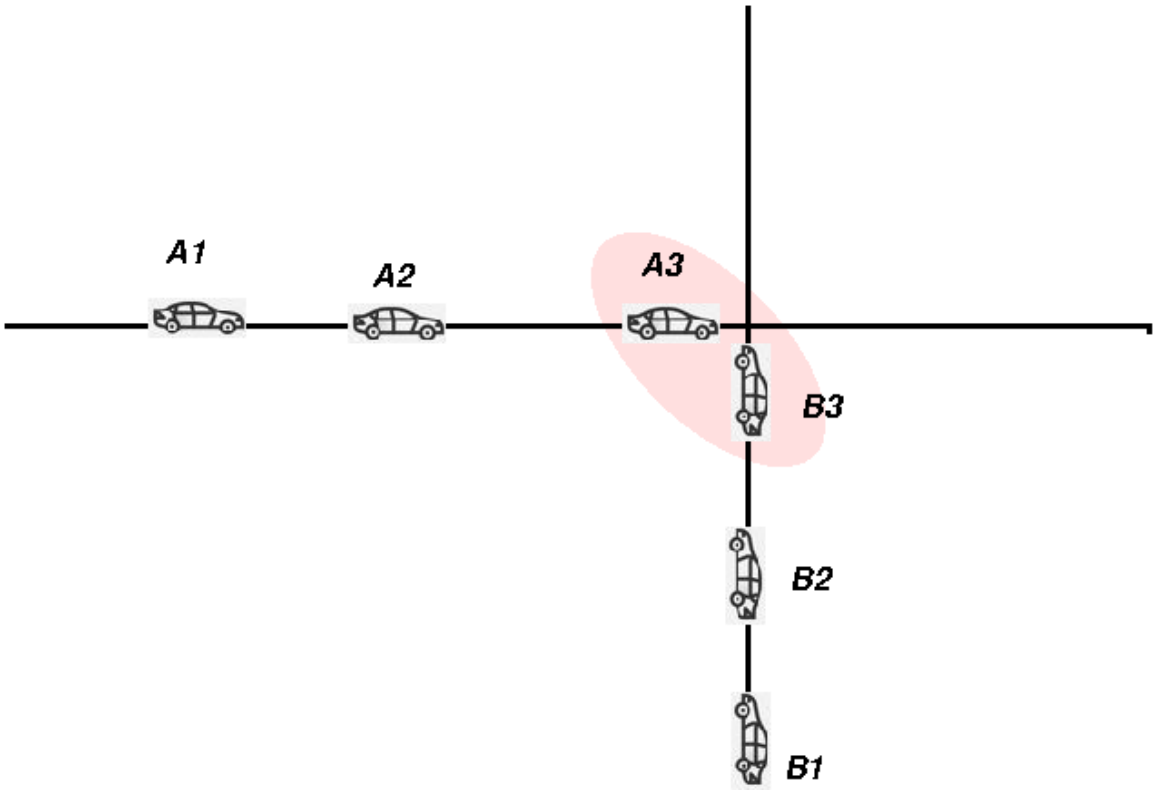


Figure 3.8. Corner Collision

Note that since S_{ba} will always be greater than zero, the only point of collision is when $S_{ba} = 0$. These cases of collision can happen during merging or switching lanes.

- **Case 3: Corner Collision**

  Figure 3.8 and 3.9 depicts Case 3, which is a special case of Case 2 where $\theta = 90°$. The distance between car A and B at time t is, therefore:

$$S_{ba} = \sqrt{(\Delta x)^2 + (\Delta y)^2} = \sqrt{(x_a)^2 + (y_b)^2}$$

  These collision cases usually happen at intersections between vertical and horizontal traffic.

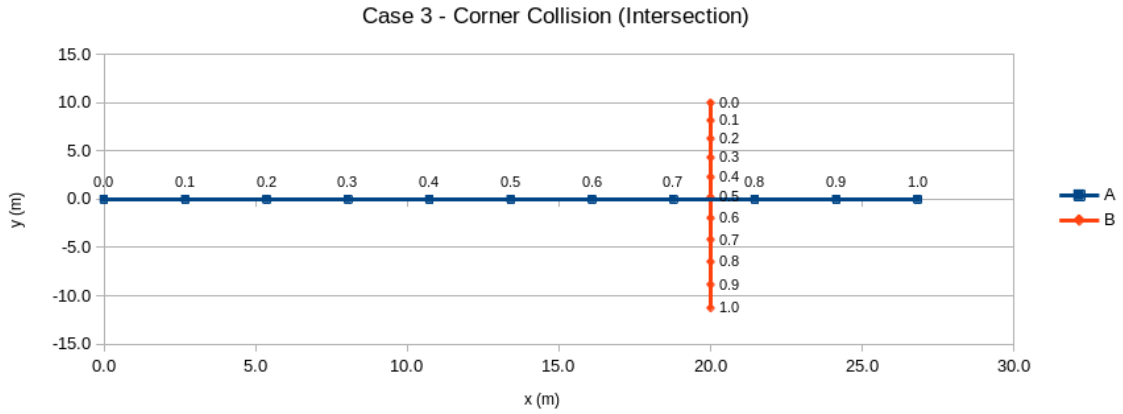


Figure 3.9. Corner Collision in Action

3.2.2 Generalized Formula

From the above section, we can obtain the point of collision when $S_{AB} = 0$ using our generalized formula below with cars A and B respectively at positions (x_A, y_A) and (x_B, y_B):

$$S_{AB} = \sqrt{(\Delta x)^2 + (\Delta y)^2} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

We can then apply this formula to predict the minimum distances between two cars, enabling us to detect potential collisions.

To calculate the position prediction of a car, we need the car's maximum acceleration (a^+) and deceleration (a^-), as well as t_p, the amount of time to predict in advance. By knowing the car's current position (x_0, y_0), speed v_0 and direction measured in θ, we can derive the following formulas below to calculate the probable locations of the car in between the points $({x_A}^+, {y_A}^+)$ and $({x_B}^-, {y_B}^-)$.

$$\begin{cases} {x_A}^+ = x_0 + v_0 t_p \cos\theta + \frac{1}{2} a^+ {t_p}^2 \cos\theta \\ {y_A}^+ = y_0 + v_0 t_p \sin\theta + \frac{1}{2} a^+ {t_p}^2 \sin\theta \\ {x_B}^- = x_0 + v_0 t_p \cos\theta + \frac{1}{2} a^- {t_p}^2 \cos\theta \\ {y_B}^- = y_0 + v_0 t_p \sin\theta + \frac{1}{2} a^- {t_p}^2 \sin\theta \end{cases}$$

4 Simulation

4.1 Generating Cases

The test cases are generated by the Python script `velocity_generate.py` into a CSV file and the visualization of the setup is shown in Figure 4.1. This specific script generates the velocity data for two cars near a highway ramp, with car 1 (in orange) already on the highway and car 2 (in blue) merging onto the highway. In 2 seconds, car 1 slows down from 65 mph to 55 mph when car 2 pulls in front of car 1 from the ramp, accelerating from 60 to 70 mph. The functions below describe the velocities of the two cars between $t = 0s$ to $t = 2s$ with a sampling resolution of 0.02 seconds:

$$v_1 = \begin{cases} 65 - 10t, & 0 \leq t < 1 \\ 55, & 1 \leq t \leq 2 \end{cases}$$

$$\theta_1 = 0, 0 \leq t \leq 2$$

$$v_2 = 60 + 5t, 0 \leq t \leq 2$$

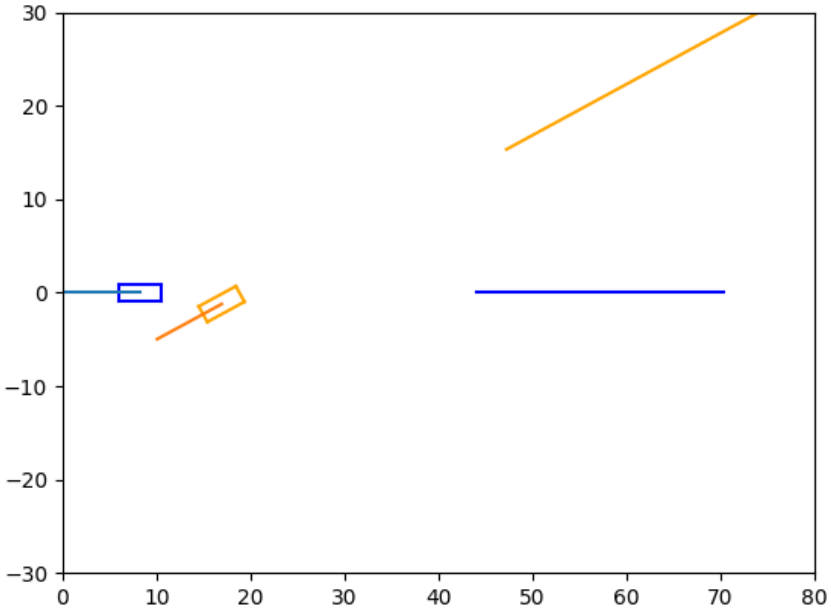


Figure 4.1. Predictions of cars 1 and 2 at $t = 0$

$$\theta_2 = \begin{cases} 0.5, & 0 \leq t < 0.4 \\ 0.3, & 0.4 \leq t < 0.6 \\ 0.1, & 0.6 \leq t < 1 \\ 0, & 1 \leq t \leq 2 \end{cases}$$

4.2 Results

By running the Python script `run.py` on the CSV generated according to the above functions, we obtain an animation of two cars' trajectories presented in Figure 4.1.

After simulating all cases in 2 seconds with a time resolution of 0.02 seconds, the cars have moved to their new positions shown in Figure 4.2. In the figure, the predictions of the cars at $t = 0s$ are overlapped as a lighter shade with the positions

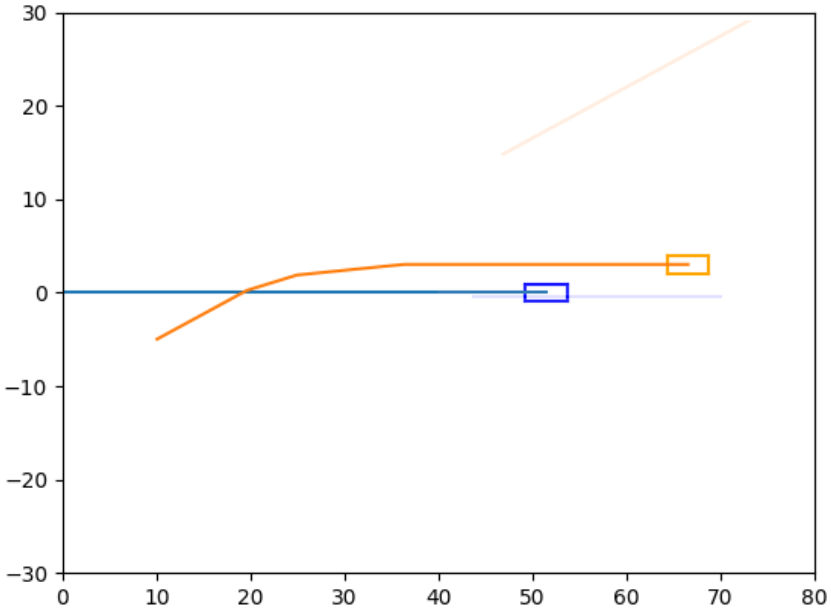


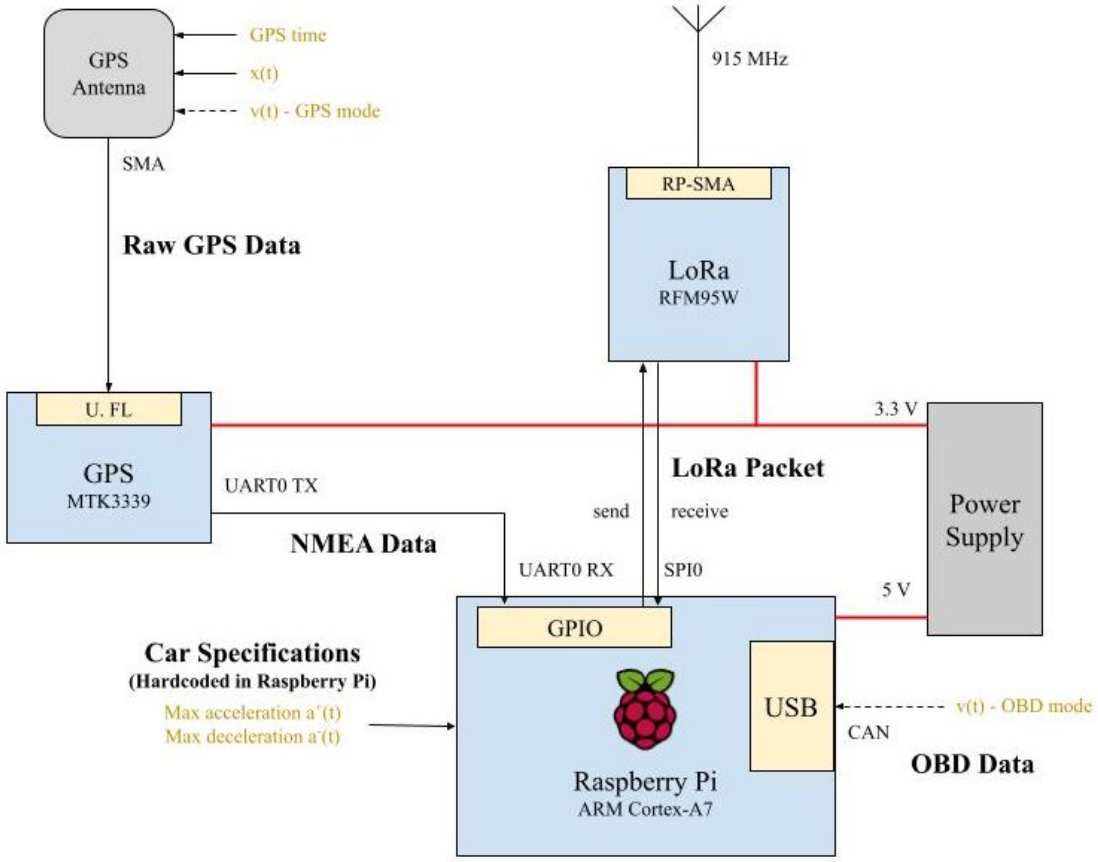
Figure 4.2. Predictions of cars 1 and 2 at $t = 2$

of the cars at $t = 2s$, showing that the actual position of car 2 at time $t = 2s$ is indeed in the range of the prediction at $t = 0s$ on a straight line.

5 Hardware Architecture

The software implementation plan in the previous shows the programming between the different components involved in obtaining vehicle movement data, calculating predictions, and communicating through the radio channel. This section further describes the system implementation decisions leading to a prototype for testing the performance of the program. Figure 5.1 shows the hardware architecture of the communication module for each testing vehicle.

Figure 5.1. Hardware Architecture of Model Implementation



5.1 Peripheral Communication

- **Serial UART**

  The serial universal asynchronous receiver-transmitter (UART)[27] is a communicating method between the computer and peripheral devices through the serial interface. The two devices communicate by configuring the same fixed baud rate for data transmission through the Tx and Rx pins. In this project, the Raspberry Pi communicates with the GPS module and the ELM327 through two serial UART lines with baud rates of respectively 115200 baud and 38400 baud.

- **SPI**

  The serial peripheral interface (SPI)[22] is a serial interface with a synchronous clock signal to conduct the data transfer between the master and slave devices on the interface. The master and slave coordinate through the Master In Slave Out (MISO) and Master Out Slave In (MOSI) lines to form a ring buffer between the two devices. During each clock cycle, the master shifts one bit to the slave while the slave shifts one bit to the master. This process guarantees a bi-directional data transfer on the SPI interface.

  In this project, the SPI interface GPIOs are connected to the LoRa radio module. Our radio module also uses the DIO pin, which is programmed to receive interrupts from the LoRa module during the LoRa radio transceiving process.

- **CAN**

  The Controller Area Network (CAN) bus is the interface that communicates with the OBD2 port on the car. In this project, the CAN interface is consolidated into the ELM327 board[9] that redirects the CAN bus traffic to serial UART for the Raspberry Pi to access through USB. The CAN-specific commands are shielded behind the ELM327 implementation, and therefore require the python-OBD library to extract the data from the ELM327 through the serial bus.

5.2 Raspberry Pi

The Raspberry Pi 2 comes with the ARM Cortex-A53 processor, which has the adequate computing power to perform straightforward arithmetic operations such as trigonometric functions and square roots in Python[11]. As for peripheral communication, the Raspberry Pi supports multiple GPIO pins and interfaces such as SPI, serial UART, and USB. Our choice to adopt the Raspberry Pi in our hardware design was for its simplistic setup for communication between different hardware interfaces and the software compatibility with Python, therefore gaining access and support to a wide selection of Python libraries.

5.3 GPS Module

This project adopted the MTK3339 GPS module[31] as the source for GPS data. The chip has an update rate of 5 Hz and requires a GPS fix in order to output GPS data to the Raspberry Pi. The MTK3339 GPS chip can find a stable GPS fix either through the chip's own internal GPS antenna or through an external GPS antenna connected to the U.FL socket. During our tests, the GPS module frequently fails to receive a GPS fix with either antenna and hindering the data collection process.

5.3.1 Drawbacks of GPS Data

The GPS module is a convenient approach to obtaining position and velocity information for each node without widely deploying infrastructure that provides position coordinates. However, the provided GPS position data proved to be unreliable because of reasons including but not limited to the following listed:

(1) GPS modules are not precise enough to detect all movements. GPS data usually comes with significant position inaccuracy especially with stationary and low-speed targets, with errors up to around $\pm$ 15 meters and 2 km/h.
(2) The GPS module requires a GPS fix in order to provide position data. Weather and environmental conditions have a huge impact on the ability of the module to receive data.
(3) GPS information does not get updated as frequently as desired. The sampling rate for commercial GPS modules is usually 5 Hz and caps at 10 Hz at best.

Sampling Rate.

The sampling rate of the GPS in this project is 5 Hz. The best-case scenario of the model hardware operating at this speed is when the following steps are executed in less than 0.2 seconds before UART updates the current GPS data:

(1) Format and broadcast the current GPS data as a radio packet.
(2) Receive the incoming radio packet from another node.

(3) Calculate the distance between the two nodes and the location bounds of the nodes in the next certain unit of time.

Assuming that timeout does not occur between the two radio modules, these steps can be executed almost instantly. Therefore, the GPS sampling rate essentially bottlenecks each update interval at 0.2 seconds. Since velocity is the first derivative of position with respect to time, and acceleration is the first derivative of velocity with respect to time, accurate velocity information will now require at least 0.2 seconds to be calculated, while acceleration will require a whopping 0.4 seconds to be calculated.

In this specific test case when radio timeouts occur, the average time between position points are around 0.37 second. Obtaining velocity information will now require 0.37 seconds and acceleration will require a whopping 0.74 seconds. By intertwining multiple 0.2-second GPS sampling times and radio timeouts, it is highly probable to have missed a few seconds of prediction at a time before the packet conflict is resolved.

Radio Drop-out.

Our test case (available in the Test Cases section as Figure 7.1) shows the extent of the radio signal ending at around 50 meters. The radio signal range could be extended by tuning up the spreading factor so that each node obtains wider and more accurate radio bit signals during transmission. This will theoretically decrease the number of bits able to be transmitted in one second, but so far the radio data packets are short enough for this effect to be negligible in practice.

5.4 OBD Sensor

On-board diagnostics (OBD)[26] is a CAN interface for external devices to interact with the computer system in the car. The onboard data is usually accessible through the OBD2 port of the car. This project chooses the ELM327 chip for retrieving OBD2 data from the vehicle. The ELM327[9] is an OBD to RS232 interpreter that delivers OBD2 signals through the serial UART channel. The ELM327 enables the

Raspberry Pi to fetch real-time speed data from the vehicle directly. With the help of the Python OBD module, the Raspberry Pi could interpret the received serial data and begin collecting speed information on the fly.

5.4.1 Alternative to GPS Data

An alternative we are looking at is using the OBD sensor information provided by the car's OBD2 port. This port runs on the CAN protocol and enables car users to retrieve basic information about the vehicle while the engine is running and identify potential problems with the various problems with the car.

OBD2 is a decent alternative to GPS signals since the OBD2 data will be readily available as long as the car is up and running and therefore eliminating the stability issue with our former GPS implementation.

5.4.2 Drawbacks of OBD Data

Although OBD2 conveniently provides information about the car position and speed, there are a few limitations of using the OBD2 data:

- OBD2 provides vehicle speed to the nearest 1 $\mathrm{km\,h^{-1}}$ and the odometer to the nearest 0.5 km. Speed accuracy still needs to be tested for usability as the OBD2 data rate ranges from 10.4 to 41.6 kbps and the OBD2 update rate limit is around 20 queries per second. However, the odometer will be less used for measuring vehicle displacement since changes within 500 meters do not reflect on the odometer.
- The OBD2 bus does not provide yaw sensors. This data needs to be provided from elsewhere.

5.5 Radio Communication

This project chooses the LoRa module, a low-power radio module operating on the 900 MHz frequency band, for transmitting position and speed data between vehicles. The LoRa module is powered by the SX1276 LoRa board[28] that contains

the HopeRF RFM95W chip[13] and is proven to be stable and reliable for data transmission through radio. The LoRa module connects to the Raspberry Pi through the SPI interface.

FCC allocates the 900 MHz radio band to amateur radio and industrial, scientific, and medical (ISM) equipment. The range of the band is from 902 MHz to 928 MHz and permits unlicensed low-powered devices to access the band. (Adapted from Wikipedia) The LoRa radio module operates on the 900 MHz radio band. For this project, the module frequency is set to 920 MHz to avoid packets on the default 915 MHz[30].

6 System Prototype

6.1 Data Sources

6.1.1 GPS Data

The GPS module provides geolocation information for GPS receivers to identify its latitude and longitude information. After obtaining a signal fix from the GPS satellites in view, the GPS receiver is then able to generate GPS data strings and output the strings in NMEA data format through the serial UART interface. Each line of GPS data begins with the "$" character followed by the NMEA sentences that indicate the data type the line provides.

In order to conveniently sort out the current data, the project adopted a self-developed multi-threaded Python driver to parse the NMEA data streams outputted from the GPS chip for this project. To get the position and velocity information from the GPS module, our driver looks for NMEA strings with the sentence prefixes "$GPGGA" (or "$GNGGA") and "$GPVTG" (or "$GNVTG") and extract relevant data from the comma-separated values proceeding the NMEA sentences[5]. To asynchronously update the GPS data in the background, the prototype software runs a separate thread after initialization that constantly replaces the current GPS data object with the latest data received from the GPS module. This multi-threaded implementation guarantees that the driver can instantly provide a response whenever the main Python program calls for the most recent GPS data without having to undertake the polling and string parsing process in the foreground.

6.1.2 OBD Data

To access the data coming from the OBD2 port CAN bus, the following OBD PIDs (parameter IDs) relevant to our algorithm input are identified and listed below:

- `0x0D` - Vehicle speed ranging from 0 - 255 $\mathrm{km\,h^{-1}}$ with precision to $1\ \mathrm{km\,h^{-1}}$.
- `0xA6` - Odometer ranging from 0 - 429,496,729.5 km with precision to $0.5\ \mathrm{km}$. The odometer gives information of the car mileage since manufactured, which is measured by counting the number of rotations and multiplying it by π times the tire's diameter.

In our project, the ELM327 board and the python-OBD library[12] simplify this data-extracting process from the CAN bus.

6.2 Vehicle Communication

6.2.1 pyLoRa

This project initially used the readily available Python library for the LoRa - pyLoRa[23] (originally named pySX127x) - to attempt communication with the LoRa radio module. However, the library was not able to register the LoRa module consistently. Therefore, this project implements a bare-bone LoRa library that directly reads and writes to the LoRa registers and manually flips the IRQ flag with Python.

Our Python implementation of SPI communication to the LoRa module on the Raspberry Pi is able to send (Tx) and receive (Rx) GPS data in raw byte format. When the LoRa switches to Tx mode, the LoRa hardware sends out the GPS data written into the FIFO pipeline, and the process exits by detecting an IRQ. In Rx mode, the LoRa module listens for an IRQ and returns the bytes in the FIFO pipeline to the main Python function. In the case when Tx and Rx modes fail to see an IRQ, the implementation also includes timeout values to terminate the current mode and retry. The section *Radio Signal Collision Avoidance* further talks about how timeouts benefit the establishment of a reliable transmission channel between nodes.

The Python implementation also includes the two-way encoding and decoding of ASCII strings into byte strings for packing outbound local GPS data to the LoRa module and extracting inbound GPS data from the other vehicle.

6.2.2 Main Program

In the main Python function of our software implementation, the program first calls the GPS driver to initialize the serial UART channel with the GPS module and then initializes the LoRa transceiver through SPI. The program then enters an infinite while loop that breaks only when there is a keyboard interrupt. Within the while loop, the program broadcasts out the most current data inquired from the GPS driver through the LoRa module and waits for an incoming string from the LoRa module. If the incoming data is not garbled, then the program continues to test whether the time strings between the two signals are within the set threshold to ensure that the data from the other car is also processed and sent out in real-time.

Below is a code snippet from the main Python program.

```
def main():
   # Initialize the GPS driver
    mtk33x9 = MTK33X9()
    mtk33x9.ser_init(config.dev_path)

    # Initialize LoRa module
    lora = LoRa()

    # Initialize data buffer for incoming GPS data from other vehicle
    rx_data = MTK33X9_data()

    try:
        while (True):
         # Get current data from the GPS driver and immediately send it out
            current_data = mtk33x9.get_current_data()
            if (current_data.is_complete()):
                lora_send_data(lora, current_data)

            time.sleep(0.1)

            # Attempt to detect incoming data from the other vehicle
            try:
                incoming_data = lora_receive_data(lora)
```

```
                if (incoming_data.is_complete()):
                    rx_data = incoming_data
                    print('Received GPS information: ')
                    print_gps_info(rx_data)
            except TimeoutError:
                print('Timeout reached!')
            except IndexError:
                print('Current data received not valid!')

            time.sleep(0.1)

            # Test whether time difference of both sets of data is less than
                the threshold
            # If less than threshold, mark as real-time
            if (current_data.is_complete() and rx_data.is_complete()):
                current_prediction = Prediction(current_data, rx_data)
                if (current_prediction.is_realtime):
                    print('Distance: ' + str(current_prediction.distance) + '
                        m')

    except KeyboardInterrupt:
        mtk33x9.ser_stop()
        GPIO.cleanup()

if __name__ == "__main__":
    main()
```

By continuously running this while loop, this program will be able to output the distance between the two LoRa nodes as long as the two nodes are actively transmitting real-time data.

6.2.3 Radio Signal Collision

Our program implementation is a primitive attempt to transmit data across vehicles and does not have a built-in signal collision avoidance system such as exponential backoff to mitigate multi-node transmission. The LoRa module will attempt to immediately send out the first available GPS data whenever the main program is launched and the drivers initialized. Therefore, the program sometimes experiences the clashing of radio signals when the two vehicles launch the main program too closely to each other.

In order to prevent radio signals from clashing again in the same channel, our collision avoidance algorithm will need to absolutely guarantee that one node is in Tx mode while the other one is in Rx mode. A collision occurs when two radio signals clash at the same time. Each node can then use this information to learn about each other's timing. In fact, this information implies that the two radio modules have coincidentally synced up an absolute timing at the moment the clash occurs. To avoid another clash of radio signals, both nodes cannot use the same timeout until attempting to re-transmit their GPS data.

The above tells us that by assigning a different timeout value for each node, we can now ensure that one of the nodes will switch to Tx mode during the timeout difference earlier than the other node, which is still in Rx mode. With this timeout difference, the two vehicles are able to communicate over the radio with a considerably reliable TDMA-like timing mechanism.

In addition, the ratio between the two nodes' timeout is also important in the probability of future collisions. In the case of transmitting through an unstable radio channel, a timeout ratio of $0.2 : 0.1$ is likely to clash more since the long-timeout node will likely clash on its first attempt with the node with the short-timeout node on its second attempt.

Figure 6.1 depicts two nodes with timeout values of 0.4 seconds and 0.7 seconds. The timeout ratio between the two nodes is $4 : 7$, with one being a multiple of 2 and the other one an odd prime number. This ratio gives the two nodes an adequate 9 tries within 2.8 seconds (the least common factor of 0.4 seconds and 0.7 seconds) to resolve the signal collision. Therefore, these are the timeout values that are hard-coded in all test cases presented in this project.

Figure 6.1. Concept of using Timeout for Signal Collision Avoidance

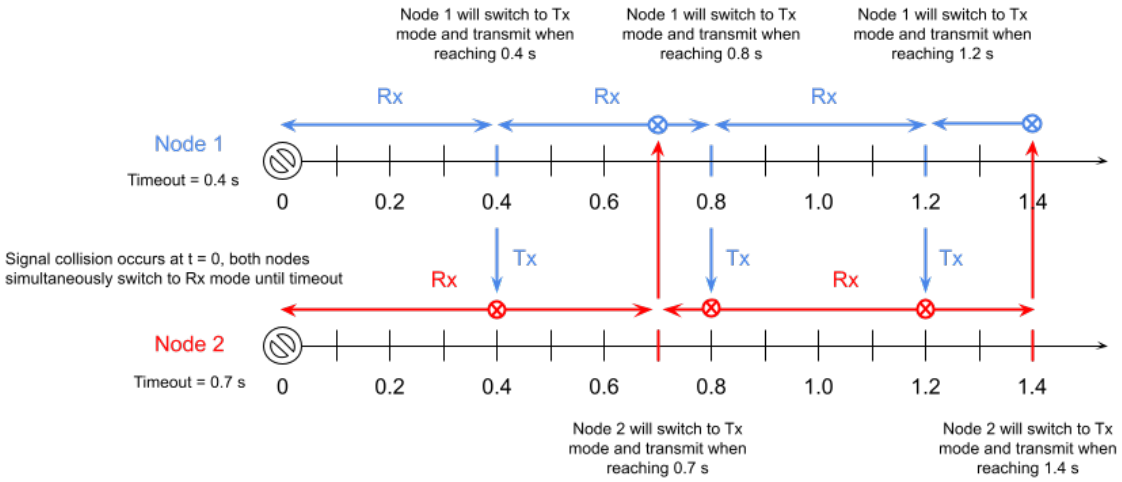
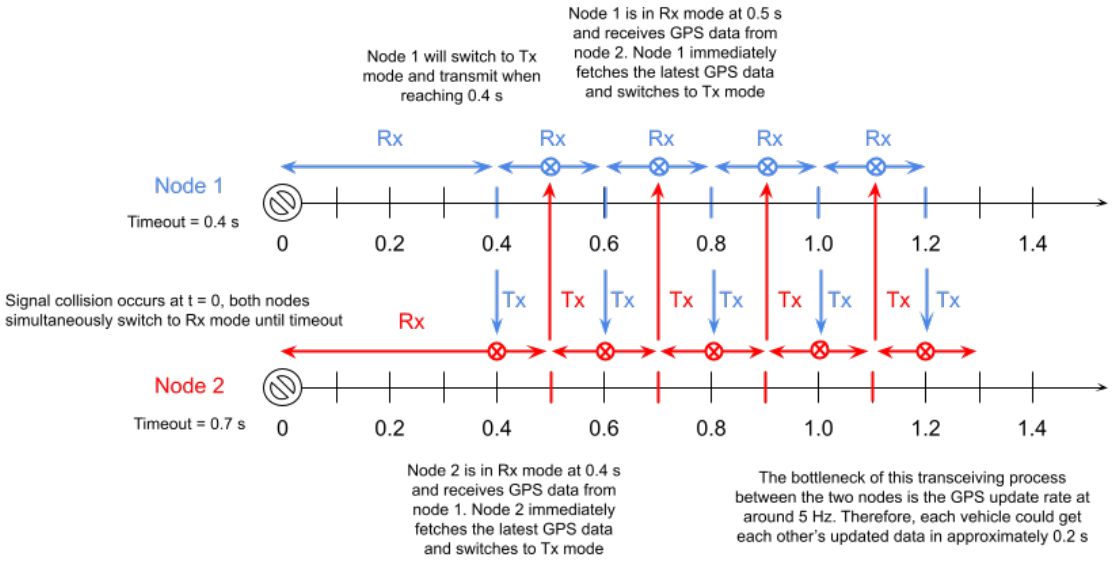


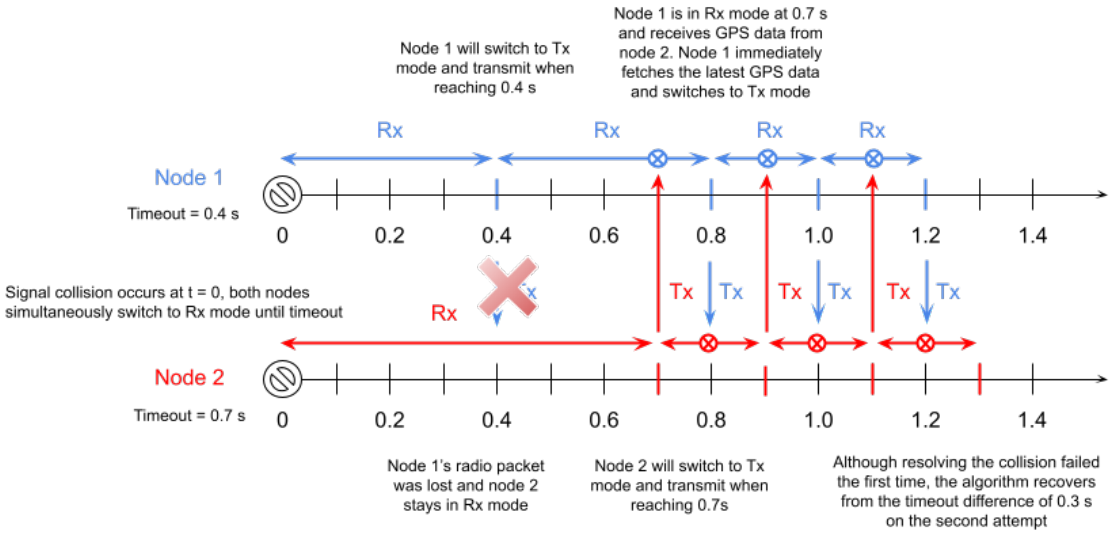
Figure 6.2 depicts the process of the radio signal resolving the clash and starting the continuous communication process within the radio channel. After a radio signal collision occurs at $t = 0s$, both nodes switch to Rx mode and start a timer with varying lengths. Node 1 first switches to Tx mode and transmits at $t = 0.4s$ while node 2 is still in Rx mode. Node 2 can then stop the 0.7-second timer, calculate the prediction, complete the current while loop iteration, and finally transmit its updated GPS data for node 1 to receive. Since each node switches right back to Rx mode immediately after completing transmitting its current data, node 1 will be in Rx mode and ready to receive the data from node 2. As long as clashes and lost packets do not occur, this communication mechanism can go back and forth without interruption.

Figure 6.2. Coordination between two nodes when signal clashes



To address the case when data gets lost during radio transmission, Figure 6.3 depicts the case when node 1 fails to deliver the first GPS data that resolve the clash. Node 1 switches back to Rx mode and starts its second iteration of the 0.4-second timer while node 2 stays in Rx mode as well for another 0.3 seconds until reaching its timeout value of 0.7 seconds. At $t = 0.7s$, node 2 switches to Tx mode and transmits its GPS data to node 1, which is 0.3 seconds into its second Rx iteration. Node 1 successfully picks up the data and promptly follows up with its data, and the continuous radio communication channel is set up as well, hence making the packet loss a small upfront cost to establishing a stable data source.

Figure 6.3. Coordination when radio packets fail to reach destination



In short, this timeout collision avoidance technique shows that the transmitted data itself, in the form of short radio packets, could directly be used to coordinate data transceiving schedules. Each node only needs to focus on broadcasting its presence to the surrounding without having to consider resolving clashes since there is no need for acknowledgments to be sent back to each node. This is only a simple method to address the communication between two vehicles. In more complicated systems, the ability for each vehicle to generate its own unique timeout becomes necessary to maintain collision-immune timeout ratios such as $0.29 : 0.47 : 0.53$.

6.3 Kalman Filtering

Our project involves two data sources - the GPS and the OBD, to track the position and speed of the vehicle, each having its own measurement error. This combination of motion sensors creates differences in measuring the location of the vehicle between the data provided by the GPS and the OBD sensors. In this case, the vehicle cannot determine its own precise location by having two sets of data with

different location values. By using sensor fusion, however, the vehicle could find an approximate consensus of location between the two sensors by deciding the weight of data from each sensor in different scenarios.

6.3.1 Filter Setup

The data sources in the project include the GPS module and the OBD port directly linked to the car. Each data source has different accuracy and precision characteristics that work better in different scenarios. The lists below show the advantages and drawbacks while using these two sensors:

- **GPS**
  - The GPS module has a Slower update rate of 5 Hz and cannot accurately measure short-term distance and speed within 0.2 seconds. All speeds within the update time frame must be approximated by using the 0.2-second time frame.
  - The GPS module yields better results when measuring the absolute position on much longer distances of 50+ meters.
  - GPS fix accuracy works better with higher speeds. GPS data is noisy when vehicles are parked or moving at low speeds, as shown in Figure 6.4.
  - The GPS module is prone to lose the GPS signal/fix from time to time. Another sensor is necessary to fill in the gaps in places with weak GPS signals.
- **OBD**
  - The OBD sensor has a much faster update rate of around 30-50 Hz compared to the GPS module. All speeds are output from the vehicle in real-time, making the calculation of short-term position and acceleration much more accurate.
  - The OBD sensor yields worse results when measuring the absolute position on longer distances. The OBD data is prone to drifting as integrating the speed with respect to time induces error in calculating the vehicle's position.

- The OBD sensor works better with lower speeds. OBD data is clean at $0km/h$ when vehicles are parked, as shown in Figure 6.4. As vehicle speed increases, tire pressure and size can skew the measurement speed from the real speed.
- The OBD sensor can produce a continuous stream of OBD data readily available for parsing at very high rates as long as the OBD connection is intact.

The above infers that the combination of the GPS module and OBD sensor has complementing advantages to produce a more accurate measurement of the position. The OBD sensor can temporarily calibrate the position of the speed whenever the GPS signal is lost or noisy, and the GPS can constantly re-position the location of the vehicle whenever errors in the OBD data start to drift to the assumed OBD position.

Kalman filtering is especially useful in our case since our model also produces two data sets respectively from the GPS output and the OBD output. It is difficult to determine which sensor yields the more accurate results in different scenarios, yet the goal of our project is to find the most accurate and consistent value of the position and speed of the vehicle.

6.3.2 Project Model

The Kalman filter implemented for this project is based on changing the weights of the OBD data and the GPS data. The Kalman filter requires samples to initialize the filter so that the filter can capture the errors in either sensor as the difference. The filter then uses the sample data provided by previous iterations of the program to continuously update the filter until the filter converges to a constant matrix.

In this project, our Kalman filter chooses to fuse the distance first and then the speed since differentiating the distance is less prone to drift compared to integrating the speed.

In order to get a more accurate measurement of distance, the OBD data is computed beforehand so that the first Kalman filter can fuse the distance data between the GPS distance and the OBD distance first. The second Kalman filter will attempt

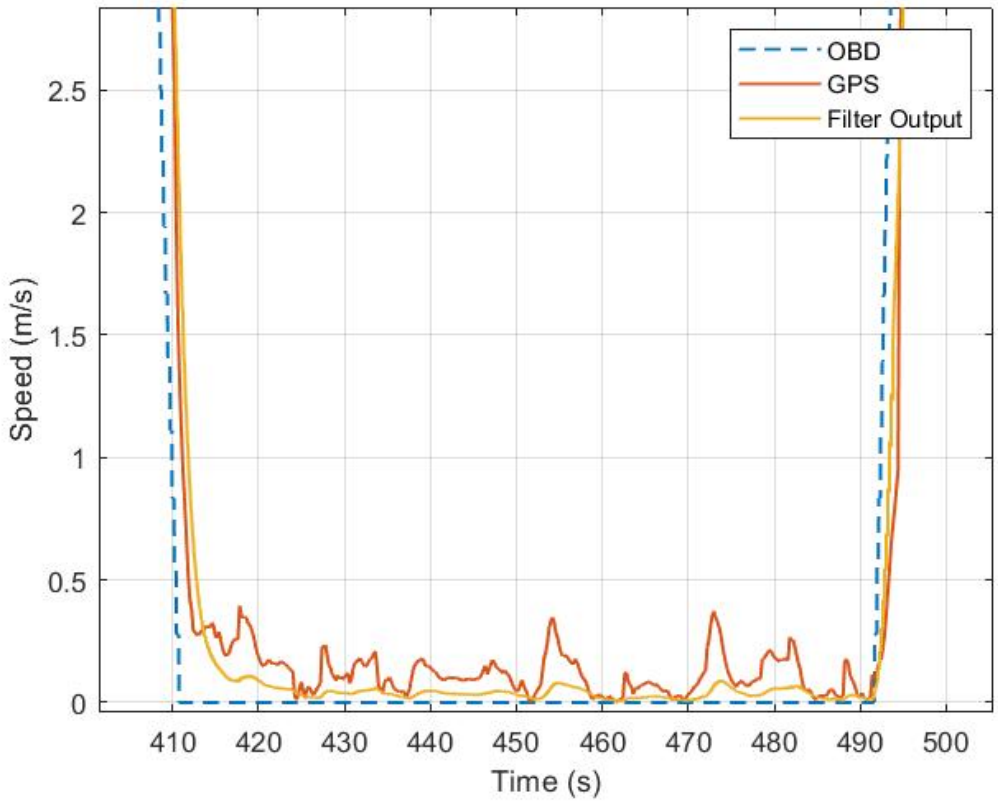


Figure 6.4. Noise at speed = 0

to fuse the difference between the speed output of the first Kalman filter (which includes OBD data and GPS data) with the OBD speed, as the OBD sensor is able to provide a more accurate instantaneous speed compared to the GPS module.

6.3.3 Kalman Equations

There are two steps in When defining the Kalman filtering algorithm for our project, there are two steps involved in each iteration - updating the next base value for prediction with the incoming data and predicting the next iteration to produce the next base value. In our code, the prediction function is written below, with F as an arbitrary and adjustable constant:

```
function [r, P] = predict(r, dt, P, Q)
```

```
    F = [1];
    r = F*r;
    P = F*P*F' + Q;
end
```

The update function calculates the Kalman gain for each iteration. Each iteration will update the Kalman gain according to the incoming data until it converges to a certain value, making subsequent data less significant in influencing the prediction values. The update function is written below:

```
function [r, P] = update(r, z, P, R)
    K = P/(P + R); %Kalman gain

    r = r + K*(z - r);
    P = (eye(1) - K)*P*(eye(1) - K)' + K*R*K';%P - K*P;
end
```

A few parameters to flush out in the following progress notes. In the parameters listed below, Q is the prediction noise, R is the measurement noise covariance for each sensor, and P is the prediction uncertainty:

$Q = 0.1$	System noise
$R1 = 2000$	R for OBD sensor
$R2 = 100$	R for GPS module
$P = 1$	Prediction uncertainty

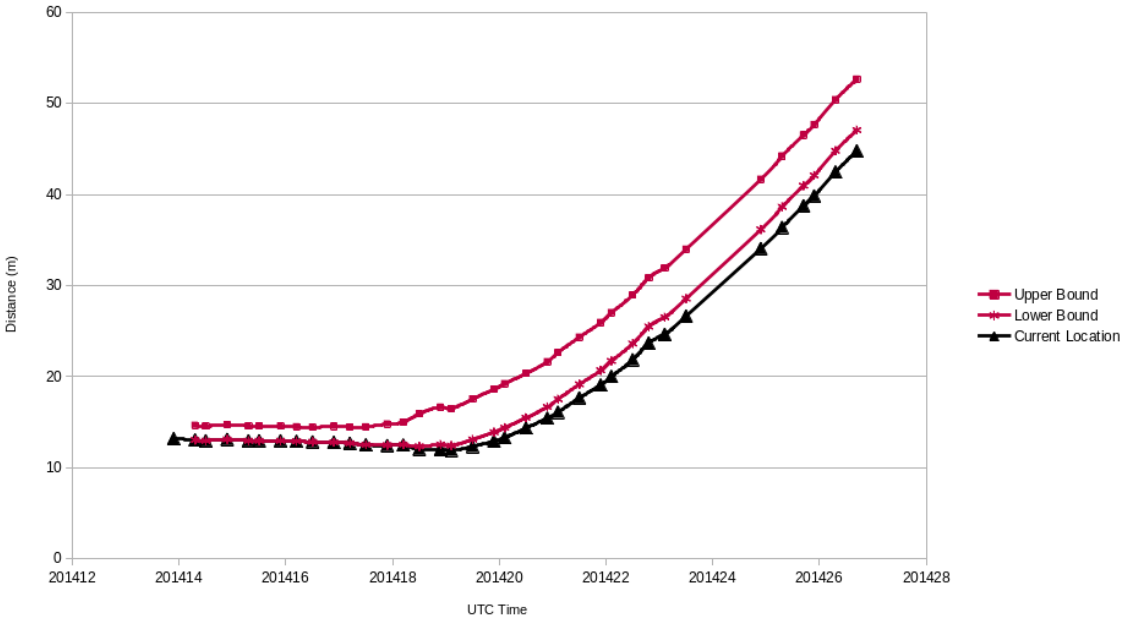
Our first design of the Kalman filter is to find consensus by using the time duration of the vehicle. The farther the vehicle is from the starting point, the more important GPS data becomes.

7 Test Cases and Results

7.1 GPS Test Case

Our first test case involves two nodes, one installed on the side of the street and the other one installed inside a Toyota Corolla LE with a 0-60 time of approximately 8.6 s (3.119 m/s^2) and 60-0 braking distance of 135 $ft.$ (-8.742 m/s^2). The car drives away from the stationary node and the node in the car collects the data presented below. Figure 7.1 depicts the recorded results of the distance between the two nodes and the implied position bound in the following second. This bound is calculated from the maximum acceleration when flooring the gas (3.119 m/s^2) and maximum deceleration when brakes are fully applied (-8.742 m/s^2), both of which are estimated from the given 0-60 and 60-0 values. Note that in the figure at 20:14:24 UTC, the radio lost connection for approximately 1 second.

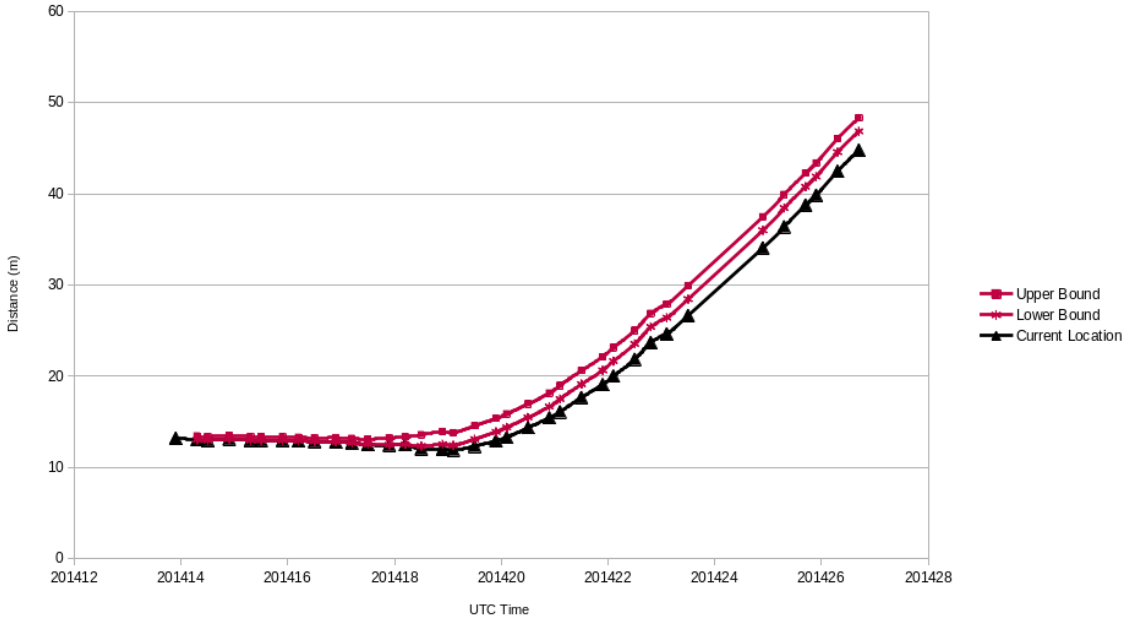
Figure 7.1. Real data and prediction bounds derived from data



7.1.1 Prediction

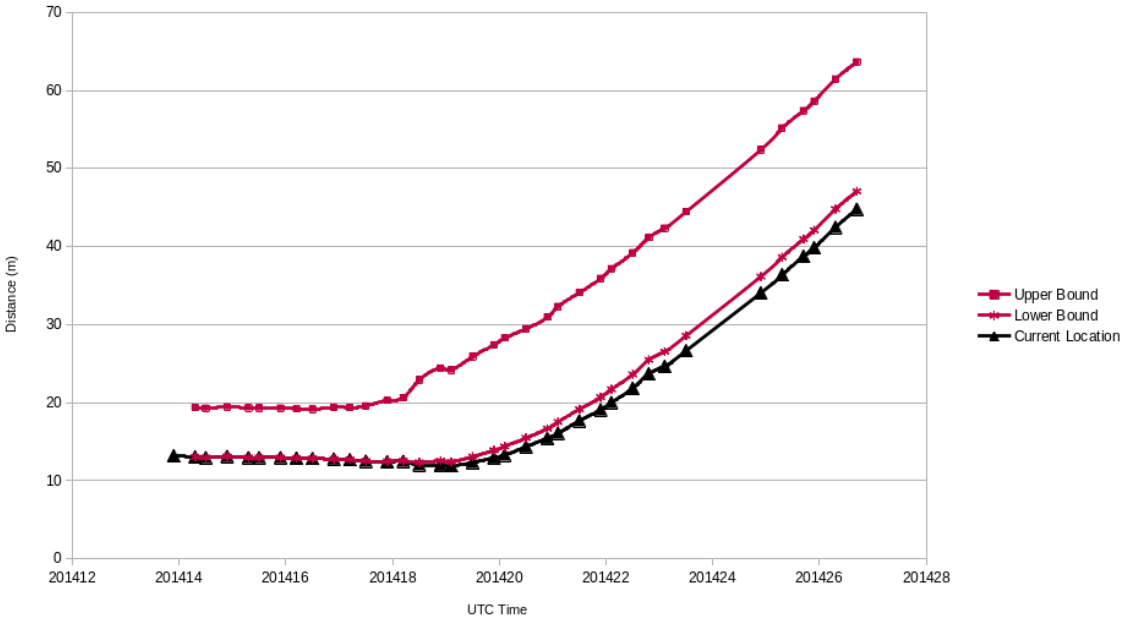
So far, our program is able to predict the upper and lower bounds of the car's location one second ahead, as shown in Figure 7.1. This one-second prediction bound could be changed in order to account for different time frame requirements, as shown in Figures 7.2 and 7.3 below.

Figure 7.2. Predictions 0.5 second ahead



As this prediction time frame decreases to around 0.5 seconds, the prediction becomes less valuable since the driver will have less time to react to the upcoming scenario. Moreover, prediction accuracy will decrease as the time frame approaches the GPS sampling rate.

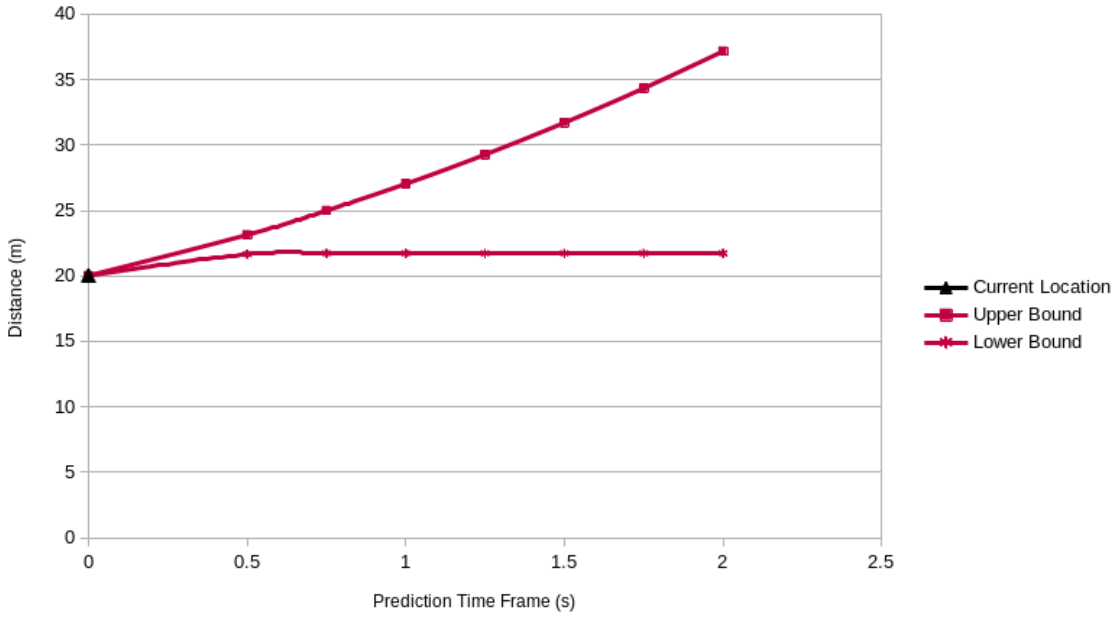
Figure 7.3. Predictions 2 seconds ahead



However, as this prediction time frame increases to 2 seconds, the upper and lower bounds become much farther apart. The information provided by the prediction then becomes too obvious, which can also void the usefulness of the prediction. To conclude, we can infer that there is a trade-off between location accuracy and reaction time to the prediction.

Figure 7.4 shows the prediction bounds for various prediction time frames at 20:14:22.1 UTC.

Figure 7.4. Predictions made for different time periods in advance at 20:14:22.1 UTC



7.2 OBD Test Case

In Figure 7.5, the Kalman filter first weighs towards the OBD data (shown as Sensor input 1) to capture distance in the first 20 seconds. After around 30 seconds, the Kalman filter starts to factor in the GPS data (shown as sensor input 2) more heavily over time.

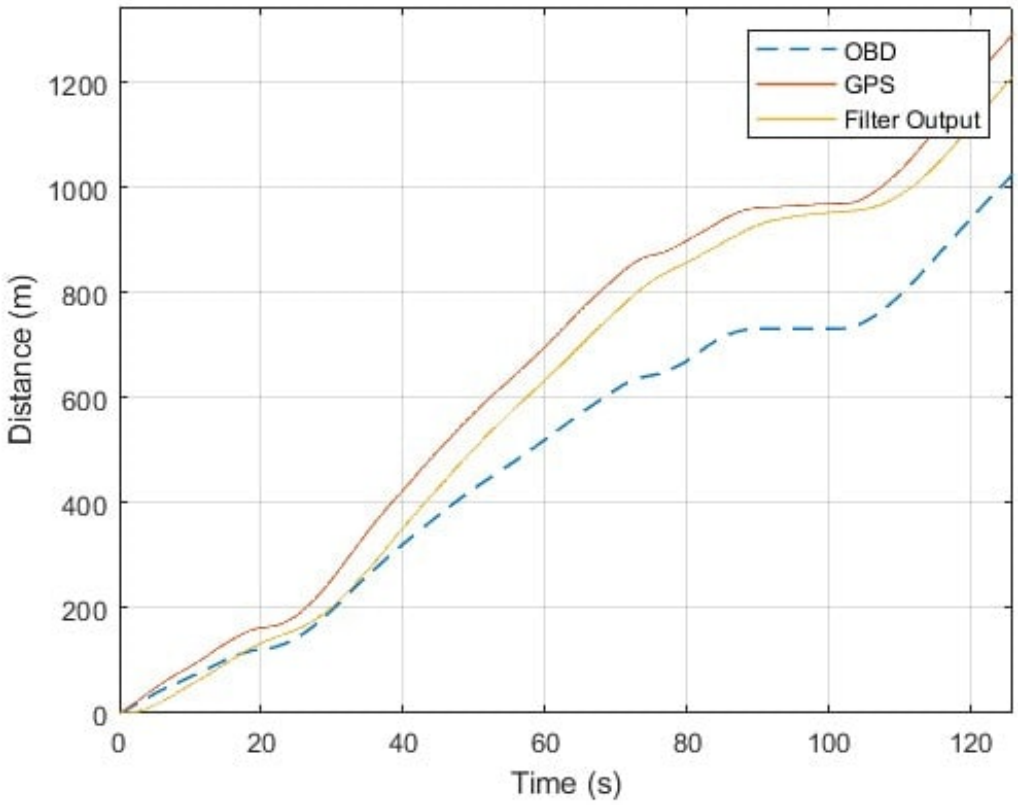


Figure 7.5. Filtering results of Distance (m)

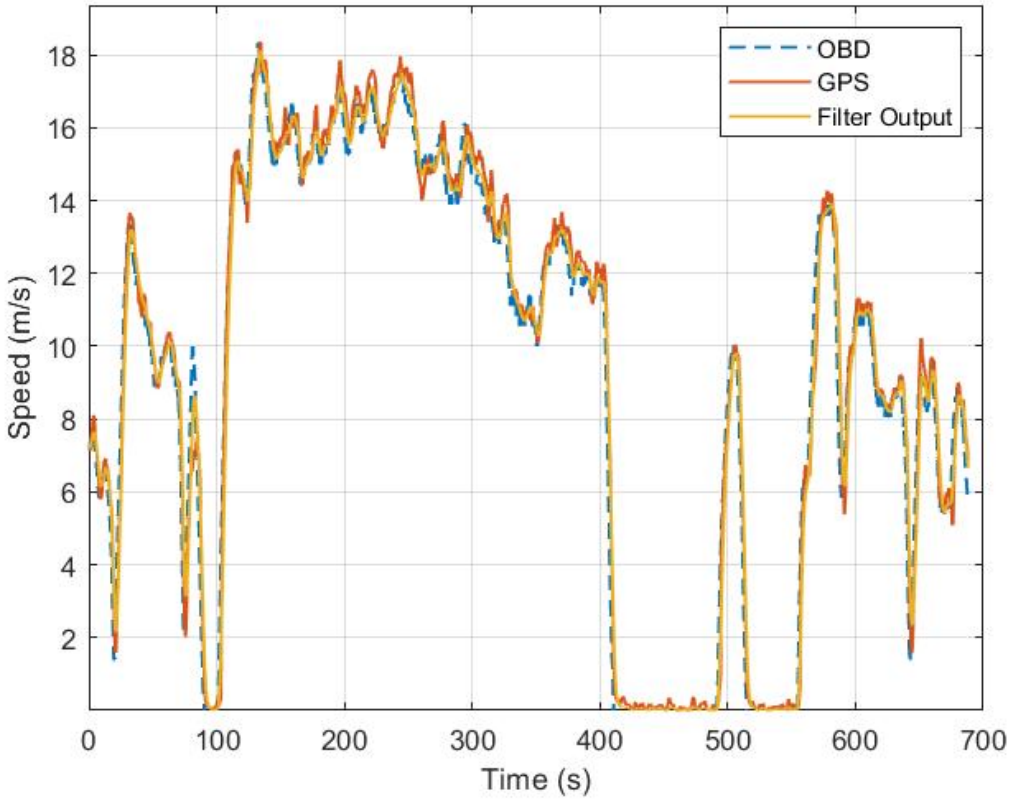


Figure 7.6. Filtering results of Speed (m/s)

8 Conclusions

This paper proposed the concept and methodology of transmitting real-time data through radio for car communication, as well as fusing sensor data between GPS and OBD to provide accurate data for the connected car architecture. A prototype of the software algorithm and hardware design is implemented and tested, and results show accurate calculations of car position range in the following second.

The successful results of the research prove to ensure greater safety when autonomous sensors cannot detect potential hazards by providing wireless beacons with GPS data as a fall-back method to avoid traffic collisions.

In addition, by fusing OBD information with the GPS data, real-time accuracy improves as the OBD port provides fast updates on car velocity, which can smooth out the measurement errors of GPS data on the fly.

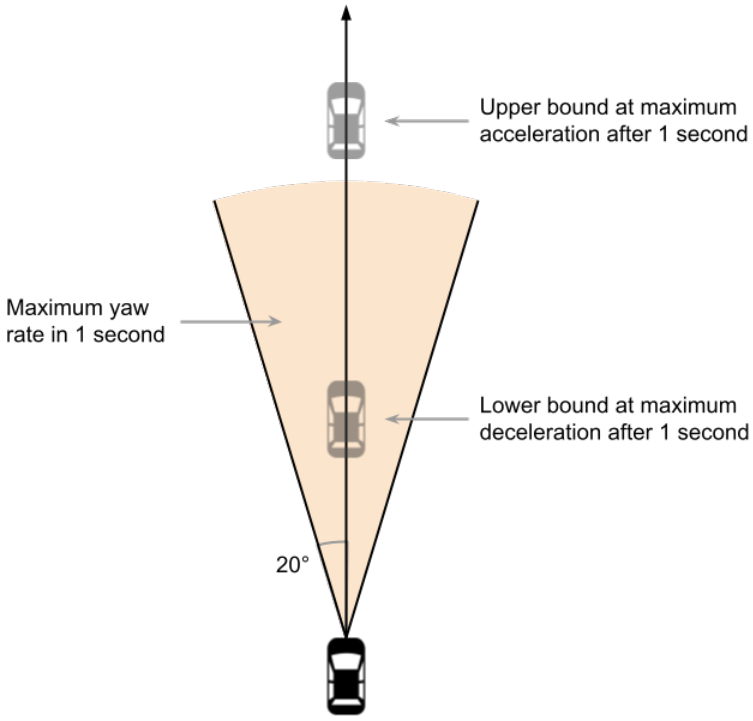
Ultimately, the software ideas and hardware architectures presented in this paper could serve as a lightweight platform and test bench for developing new software algorithms that can benefit safety by leveraging existing connected vehicle technology.

9 Suggested Future Research

9.1 Adding Angle

To analyze the model in 2-D space, we can add in the control of the steering wheel. This can be done by measuring the yaw rates of the car. Typically a value of 20 deg/s is already considered aggressive at high driving speeds. We can then set a certain amount of yaw rate as the bounds for turning the steering wheel and draw out a circular sector-shaped range for the direction perpendicular to the moving direction. Figure 9.1 shows the addition of analyzing yaw rates to the model.

Figure 9.1. Addition of Yaw Rate to the Model



9.2 CSMA/CA

Carrier-sense multiple access[25] with collision avoidance (CSMA/CA) is a protocol at the media access control (MAC) layer for mitigating congested network traffic on a certain wireless communication channel. The protocol specifically uses the exponential random backoff algorithm that retries transmission at a random delayed time frame whenever collision in the channel is detected until one of the transmission nodes can get an acknowledgment of successful transmission. This project implements collision avoidance between the LoRa nodes with timeout differences so that one node can always transmit before the other whenever a collision occurs.

A more sophisticated collision avoidance mechanism such as CSMA/CA could be implemented in place of fixed timeouts.

9.3 Stacked Lane Infrastructures

Another factor to consider is that not all roads are 2-dimensional only. Some bridges, airports, and interchanges have lanes stacked on top of each other. Those scenarios will require more sources of information to identify which layer the car is located in.

9.4 Wireless Security

One of the most important challenges to overcome for the proposed decentralized framework is the security of the wireless channel the vehicles communicate in. Malicious users can block the wireless channel from being usable with frequency jamming. In addition, the framework also needs to filter out fake or illegitimate wireless beacons from interfering with the communication channel.

Appendix A

`run.py`

Below is the Python source code for `run.py`. The Python script calculates the predicted trajectories for the cars in the upcoming unit of time. The script inputs position and speed data from a CSV file `speed.csv` and outputs the calculated predictions into `results.txt`. Simultaneously, the script can also re-read the data from `results.txt` and draw a plot with the car dimensions and the predicted trajectories for visualization purposes.

```
#!/usr/bin/python3

import csv
import math
import os
import time
import threading

from termcolor import colored

import matplotlib.pyplot as plt
import matplotlib.animation as animation


# time frame of each entry in seconds
unit = 0.02

# prediction ahead of time in seconds
predict_time = 2


# maximum acceleration and deceleration of car 1 and 2 in ms^-2
# car 1 - 2020 Toyota Corolla XSE
car1_max_acc = 3.3
car1_max_dec = -9.9

# car 2 - 2020 Mercedes-Benz AMG GT
car2_max_acc = 6.9
car2_max_dec = -10.2


# initial position of the cars (at t = 0)
```

```
car1_init_pos_x = 0
car1_init_pos_y = 0

car2_init_pos_x = 10
car2_init_pos_y = -5


# the dimensions of car 1 and 2 in meters
# Toyota Corolla
car1_l = 4.6
car1_w = 1.8

# Mercedes-Benz AMG GT
car2_l = 4.5
car2_w = 1.9


# set up graph plotting
fig = plt.figure()
axes = fig.add_subplot(1, 1, 1)

# a helper function that converts miles per hour to meters per second
def mph_to_mps(speed_mph):
    return float(speed_mph) * 0.44704

# a helper function that calculates current position from previous position
    and speed
def current_position(prev_pos, speed):
    return prev_pos + speed * unit

# helper function that calculates the distance between
def dist_xy(x1, y1, x2, y2):
    return math.sqrt(math.pow((x2 - x1), 2) + math.pow((y2 - y1), 2))

# a helper function that writes the calculated results to file
def write_data_to_file(speed1_list, angle1_list, speed2_list, angle2_list):

    # assume car 1 and car 2 has the same number of data entries
    speed_list_len = len(speed1_list)

    # current position of car 1 in vector form initialized with x and y at t
       = 0
    x1 = car1_init_pos_x
    y1 = car1_init_pos_y
```

```
# current position of car 2 in vector form initialized with x abd y at t
    = 0
x2 = car2_init_pos_x
y2 = car2_init_pos_y

for i in range(speed_list_len):

    # slow down animation by a little
    time.sleep(unit)

    # open file to write line
    txt_fd = open('results.txt', "a")

    # current speeds of car 1 and car 2
    v1 = mph_to_mps(speed1_list[i])
    v2 = mph_to_mps(speed2_list[i])

    angle1 = float(angle1_list[i])
    angle2 = float(angle2_list[i])

    # update car 1 position according to v1
    x1 = current_position(x1, v1 * math.cos(angle1))
    y1 = current_position(y1, v1 * math.sin(angle1))

    # update car 2 position according to v2
    x2 = current_position(x2, v2 * math.cos(angle2))
    y2 = current_position(y2, v2 * math.sin(angle2))

    print("t={:.3f}".format(i * unit) + "s, "

            # v1 in mph
            + colored("v1=" + str(speed1_list[i]) + " mph, ", 'green')

            # v1 in m/s vector form
            + colored("v1_x=" + "{:.3f}".format(v1 * math.cos(angle1)) +
                " m/s, ", 'green')
            + colored("v1_y=" + "{:.3f}".format(v1 * math.sin(angle1)) +
                " m/s, ", 'green')

            # position of car 1 in vector form
            + colored("x1=" + "{:.3f}".format(x1) + " m/s, ", 'green')
            + colored("y1=" + "{:.3f}".format(y1) + " m/s, ", 'green')


            # v2 in mph
            + colored("v2=" + str(speed2_list[i]) + " mph, ", 'yellow')
```

```
                # v2 in m/s vector form
                + colored("v2_x=" + "{:.3f}".format(v2 * math.cos(angle2)) +
                    " m/s, ", 'yellow')
                + colored("v2_y=" + "{:.3f}".format(v2 * math.sin(angle2)) +
                    " m/s, ", 'yellow')

                # position of car 2 in vector form
                + colored("x2=" + "{:.3f}".format(x2) + " m/s, ", 'yellow')
                + colored("y2=" + "{:.3f}".format(y2) + " m/s, ", 'yellow')

                # distance between car 1 and car 2
                + "d={:.3f}".format(dist_xy(x1, y1, x2, y2)) + " m"
        )

        txt_fd.write("{:.3f},{:.3f},{:.3f},{:.3f},{:.3f},{:.3f},{:.3f},{:.3f}\n".format(x1
            y1, angle1, v1, x2, y2, angle2, v2))

        txt_fd.close()


def draw_car_dimension(car_l, car_w, car_pos_x, car_pos_y, car_angle,
    line_color):

    # locate the four edges of the car
    edge_ax = car_pos_x - car_l / 2 * math.cos(car_angle) - car_w / 2 *
        math.sin(car_angle)
    edge_ay = car_pos_y - car_l / 2 * math.sin(car_angle) + car_w / 2 *
        math.cos(car_angle)
    edge_bx = car_pos_x + car_l / 2 * math.cos(car_angle) - car_w / 2 *
        math.sin(car_angle)
    edge_by = car_pos_y + car_l / 2 * math.sin(car_angle) + car_w / 2 *
        math.cos(car_angle)
    edge_cx = car_pos_x + car_l / 2 * math.cos(car_angle) + car_w / 2 *
        math.sin(car_angle)
    edge_cy = car_pos_y + car_l / 2 * math.sin(car_angle) - car_w / 2 *
        math.cos(car_angle)
    edge_dx = car_pos_x - car_l / 2 * math.cos(car_angle) + car_w / 2 *
        math.sin(car_angle)
    edge_dy = car_pos_y - car_l / 2 * math.sin(car_angle) - car_w / 2 *
        math.cos(car_angle)

#   print("a = ("
#           + str(edge_ax) + ", "
#           + str(edge_ay) + ") b = ("
#           + str(edge_bx) + ", "
```

```
#           + str(edge_by) + ") c = ("
#           + str(edge_cx) + ", "
#           + str(edge_cy) + ") d = ("
#           + str(edge_dx) + ", "
#           + str(edge_dy) + ")"
#           )

    # Add lines to list
    lines_list = []
    lines_list.append(
           plt.Line2D((edge_ax, edge_bx), (edge_ay, edge_by),
               color=line_color)
           )
    lines_list.append(
           plt.Line2D((edge_bx, edge_cx), (edge_by, edge_cy),
               color=line_color)
           )
    lines_list.append(
           plt.Line2D((edge_cx, edge_dx), (edge_cy, edge_dy),
               color=line_color)
           )
    lines_list.append(
           plt.Line2D((edge_dx, edge_ax), (edge_dy, edge_ay),
               color=line_color)
           )

    # add lines to the plot
    for line in lines_list:
        plt.gca().add_line(line)

    #return lines_list


def draw_prediction(car_pos_x, car_pos_y, car_speed, car_angle,
    car_max_acc, car_max_dec, line_color):
    # x = x0 + vt + at^2 / 2
    car_x1 = car_pos_x + car_speed * math.cos(car_angle) * predict_time +
        car_max_acc * math.cos(car_angle) * math.pow(predict_time, 2) / 2
    car_y1 = car_pos_y + car_speed * math.sin(car_angle) * predict_time +
        car_max_acc * math.sin(car_angle) * math.pow(predict_time, 2) / 2

    car_x2 = car_pos_x + car_speed * math.cos(car_angle) * predict_time +
        car_max_dec * math.cos(car_angle) * math.pow(predict_time, 2) / 2
    car_y2 = car_pos_y + car_speed * math.sin(car_angle) * predict_time +
        car_max_dec * math.sin(car_angle) * math.pow(predict_time, 2) / 2
```

```
    #print("(" + car_x1 + ", " + car_y1 + "), (" + car_x2 + ", " + car_y2 +
        ")")

    line = plt.Line2D((car_x1, car_x2), (car_y1, car_y2), color=line_color)
#
    plt.gca().add_line(line)


def animate(i):

    # set plot axes range
    plt.gca().clear()
    axes.clear()
    #axes.set_xlim([0, 80])
    #axes.set_ylim([-30, 30])
    axes.set_xlim([0, 80])
    axes.set_ylim([-30, 30])

    # initialize array for plotting graph
    x1_arr = []
    x2_arr = []
    y1_arr = []
    y2_arr = []

    while (not os.path.exists("results.txt")):
        pass

    # reopen file for plotting graph
    txt_fd = open('results.txt', "r")
    dataArr = txt_fd.read().split('\n')

    # record the current coordinate information
    x1 = car1_init_pos_x
    y1 = car1_init_pos_y
    x2 = car2_init_pos_x
    y2 = car2_init_pos_y

    v1 = v2 = None
    angle1 = angle2 = None

    for line in dataArr:
        if len(line) > 1:
            x1, y1, angle1, v1, x2, y2, angle2, v2 = line.split(',')
            x1_arr.append(float(x1))
            y1_arr.append(float(y1))
            x2_arr.append(float(x2))
```

```
            y2_arr.append(float(y2))


    if angle1 is not None:
        draw_car_dimension(car1_l, car1_w, float(x1), float(y1),
            float(angle1), "blue")
        draw_prediction(float(x1), float(y1), float(v1), float(angle1),
            car1_max_acc, car1_max_dec, "blue")

    if angle2 is not None:
        draw_car_dimension(car2_l, car2_w, float(x2), float(y2),
            float(angle2), "orange")
        draw_prediction(float(x2), float(y2), float(v2), float(angle2),
            car2_max_acc, car2_max_dec, "orange")

    # add plot of car 1 and 2 positions
    axes.plot(x1_arr, y1_arr)
    axes.plot(x2_arr, y2_arr)

    txt_fd.close()


# self-defined thread class for writing data to file
class write_data_thread(threading.Thread):
    def __init__(self, speed1_list, angle1_list, speed2_list, angle2_list):
        threading.Thread.__init__(self)
        self.speed1_list = speed1_list
        self.angle1_list = angle1_list
        self.speed2_list = speed2_list
        self.angle2_list = angle2_list

    def run(self):
        write_data_to_file(self.speed1_list, self.angle1_list,
            self.speed2_list, self.angle2_list)


def main():

    # clear data from last execution if exist
    if os.path.exists("results.txt"):
        os.remove("results.txt")

    # open speed.csv file that contains speed for car1 and car 2
    speed_fd = open('speed.csv')
    speed1_list = []
    angle1_list = []
```

```
    speed2_list = []
    angle2_list = []

    # store the speed values in list
    reader = csv.reader(speed_fd, delimiter=',', quotechar='|')
    for row in reader:
        speed1_list.append(row[0])
        angle1_list.append(row[1])
        speed2_list.append(row[2])
        angle2_list.append(row[3])

    # initialize and start thread for writing data
    thread = write_data_thread(speed1_list, angle1_list, speed2_list,
        angle2_list)
    thread.start()

    # show plot
    ani = animation.FuncAnimation(fig, animate, interval=100)
    plt.show()

    # close all files
    thread.join()
    speed_fd.close()


if __name__ == "__main__":
    main()
```

Appendix B

`velocity_generate.py`

Below is the Python source code for `velocity_generate.py`, the Python script that generates the particular simulated test case for the prototype algorithm shown in Chapter 4. This file is created only to generate the CSV file for `run.py` to run a simulation.

```
#!/usr/bin/python3

import csv

speed_fd = open('speed.csv', "w")

unit_sec = 0.02
count = 100

speed1_list = []
angle1_list = []
speed2_list = []
angle2_list = []

# car 1 sees car 2 and slows down from 65 to 55 mph in 1 second
for i in range(count + 1):
    if i < 0.5 * count:
        speed1_list.append("{:.3f}".format(65 - i * 10 / count / 0.5))
    else:
        speed1_list.append("55.000")

    angle1_list.append("0.000")

# car 2 accelerates from 60 to 70 mph from the ramp and merges into an
    inner lane
for i in range(count + 1):
    speed2_list.append("{:.3f}".format(60 + i * 10 / count))
    if i < 0.2 * count:
        angle2_list.append("0.500")
    elif i < 0.3 * count:
        angle2_list.append("0.300")
    elif i < 0.5 * count:
        angle2_list.append("0.100")
    else:
        angle2_list.append("0.000")
```

```
for i in range(count + 1):
    speed_fd.write(str(speed1_list[i]) + "," + str(angle1_list[i]) + "," +
        str(speed2_list[i]) + "," +str(angle2_list[i]) + "\n")

speed_fd.close()
```

References

[1] Ieee standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016. doi: 10.1109/IEEESTD.2016.7786995.

[2] Iso 26262-1:2011, Dec 2018. URL http://www.iso.org/iso/cataloguedetail?csnumber=43464.

[3] Feb 2020. URL https://www.ti.com/lit/an/swra553a/swra553a.pdf.

[4] Khadige Abboud, Hassan Aboubakr Omar, and Weihua Zhuang. Interworking of dsrc and cellular network technologies for v2x communications: A survey. *IEEE Transactions on Vehicular Technology*, 65(12):9457–9470, 2016. doi: 10.1109/TVT.2016.2591558.

[5] Glenn Baddeley. Gps - nmea sentence information, Jun 2001. URL http://aprs.gids.nl/nmea/.

[6] Taajwar Bey and Girma Tewolde. Evaluation of dsrc and lte for v2x. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1032–1035, 2019. doi: 10.1109/CCWC.2019.8666563.

[7] Igal Bilik. Comparative analysis of radar and lidar technologies for automotive applications. *IEEE Intelligent Transportation Systems Magazine*, pages 2–27, 2022. doi: 10.1109/MITS.2022.3162886.

[8] Ahmed Elbery, Sameh Sorour, Hossam Hassanein, Akram Bin Sediq, and Hatem Abou-zeid. To dsrc or 5g? a safety analysis for connected and autonomous vehicles. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2021. doi: 10.1109/GLOBECOM46510.2021.9685065.

[9] Elm Electronics. Elm327 obd to rs232 interpreter - elm electronics, Jul 2016. URL https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf.

[10] Lukas Ewecker, Ebubekir Asan, and Stefan Roos. Detecting vehicles in the dark in urban environments - a human benchmark. In *2022 IEEE Intelligent*

Vehicles Symposium (IV), pages 1145–1151, 2022. doi: 10.1109/IV51971.2022.9827013.

[11] The Raspberry Pi Foundation. Raspberry pi documentation. URL https://www.raspberrypi.com/documentation/.

[12] GitHub. Getting started - python-obd, May 2019. URL https://python-obd.readthedocs.io/en/latest/.

[13] HopeRF. Low power long range transceiver module, Jul 2019. URL https://www.hoperf.com/data/upload/portal/20190801/RFM95W-V2.0.pdf.

[14] Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. An information framework for creating a smart city through internet of things. *IEEE Internet of Things Journal*, 1(2):112–121, 2014. doi: 10.1109/JIOT.2013.2296516.

[15] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

[16] John B. Kenney. Dedicated short-range communications (dsrc) standards in the united states. *Proceedings of the IEEE*, 99(7):1162–1182, 2011. doi: 10.1109/JPROC.2011.2132790.

[17] Zadid Khan, Sakib Mahmud Khan, Mashrur Chowdhury, Mizanur Rahman, and Mhafuzul Islam. Performance evaluation of 5g millimeter-wave-based vehicular communication for connected vehicles. *IEEE Access*, 10:31031–31042, 2022. doi: 10.1109/ACCESS.2022.3158669.

[18] You Li and Javier Ibanez-Guzman. Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems. *IEEE Signal Processing Magazine*, 37(4):50–61, 2020. doi: 10.1109/MSP.2020.2973615.

[19] Alexandre K. Ligo and Jon M. Peha. Cost-effectiveness of sharing roadside infrastructure for internet of vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 19(7):2362–2372, 2018. doi: 10.1109/TITS.2018.2810708.

[20] Yangyang Liu, Shuo Chang, Zhiqing Wei, Kezhong Zhang, and Zhiyong Feng. Fusing mmwave radar with camera for 3d detection in autonomous driving. *IEEE Internet of Things Journal*, pages 1–1, 2022. doi: 10.1109/JIOT.2022.3175375.

[21] Aryan Mehra, Murari Mandal, Pratik Narang, and Vinay Chamola. Reviewnet: A fast and resource optimized network for enabling safe autonomous driving in hazy weather conditions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7):4256–4266, 2021. doi: 10.1109/TITS.2020.3013099.

[22] Inc. Motorola. Spi block guide v4 - nxp, Jan 2000. URL https://www.nxp.com/files-static/microcontrollers/doc/ref_manual/S12SPIV4.pdf.

[23] PyPI. pylora | pypi, Apr 2019. URL https://pypi.org/project/pyLoRa/.

[24] Adam Rodnitzky. Sensing 201: Solid state amp; scanning lidar for mobile robots, drones, amp; autonomous vehicles, Jul 2021. URL https://www.tangramvision.com/blog/sensors-201-scanning-and-solid-state-lidar.

[25] Santhameena. S, Aninika S Adappa, K. Dhiraj Kumar, and Ananya Boyapati. Implementation of unslotted and slotted csma/ca for 802.11 and 802.15.4 protocol. In *2019 Global Conference for Advancement in Technology (GCAT)*, pages 1–7, 2019. doi: 10.1109/GCAT47503.2019.8978395.

[26] Pooja Rajendra Sawant and Yashwant B Mane. Design and development of on-board diagnostic (obd) device for cars. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–4, 2018. doi: 10.1109/ICCUBEA.2018.8697833.

[27] NXP Semiconductors. Scc2691 3 universal asynchronous receiver/transmitter (uart) - nxp, Aug 2006. URL https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf.

[28] Semtech. Sx1276 | 137mhz to 1020mhz long range low power transceiver | semtech, May 2020. URL https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE.

[29] Haoran Song. The application of computer vision in responding to the emergencies of autonomous driving. In *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, pages 1–5, 2020. doi: 10.1109/CVIDL51233.2020.00008.

[30] John Sonnenberg. Fcc part 15 ism regulations - raveon.com, May 2019. URL https://www.raveon.com/wp-content/uploads/2019/05/AN203FCC-ISM.pdf.

[31] Adafruit Learning System. Datasheet, Feb 2018. URL https://www.manualshelf.com/manual/adafruit/746/datasheet-english/page-1.html.

[32] Onur Toker and Suleiman Alsweiss. mmwave radar based approach for pedestrian identification in autonomous vehicles. In *2020 SoutheastCon*, pages 1–2, 2020. doi: 10.1109/SoutheastCon44009.2020.9249704.