Software Requirements Specification (SRS)

Team Pedestrian Safety System: Automated Pedestrian Collision Avoidance System

Authors: Matthew Tarnowsky, Ryan Burr, David Culham and Bobak Shahidehpour

Customer: Mr. David Agnew, Continental Automotive Systems, Inc.

Instructor: Dr. Betty H.C. Cheng

1 Introduction

This software requirements specification (SRS) document provides an introduction to the Automated Pedestrian Collision Avoidance System (APCAS). This section consists of the document's purpose, scope, the definitions, acronyms, abbreviations of the terms, and the overall organization of this document.

1.1 Purpose

The purpose of this document is to thoroughly describe the functionality of the APCAS. This document describes what the system does, the features of the system, and how the system behaves. This information is conveyed through diagrams and descriptive scenarios. The intended audience for this document is all stakeholders including Mr. David Agnew and his development team from Continental Automotive Systems, Inc. as well as potential developers that are interested in similar systems.

1.2 Scope

The software product that is being produced is the APCAS. This is an embedded system for an automotive system designed to work in an autonomous vehicle. The purpose APCAS is to have the vehicle avoid hitting pedestrians with minimal human interactions.

The system monitors the front path of vehicle while in motion, looking for pedestrians. Once a pedestrian is detected the system will then analyze the path between the vehicle and pedestrian to determine if a collision is imminent. If a potential collision is found, then the system will apply automatic braking to avoid the pedestrian. The APCAS also allows for the driver to input the speed of the vehicle as well as the options to enable and disable the system.

However, there are some restrictions to the system. The APCAS can only detect and analyze the information of one pedestrian at a time. This can potentially become a problem in highly populated areas. Also, the APCAS ignores pedestrians that would hit the side of the car. Therefore, the system would not brake if a pedestrian continues on his path and walks into the side of the vehicle. Finally, the only avoidance maneuver the APCAS will perform is braking. Therefore, the system will not use any steering functionality to avoid pedestrians.

The APCAS provides many benefits to drivers. The system works with autonomous vehicles, which means the car has the ability to stay in its lane, brake at intersections and remain under control during maneuvers without any interaction from the driver. The system allows the driver to input the destination he/she would like to travel to. From there the vehicle will automatically transport them there while exhibiting basic driving skills. However, the main benefit this system provides is the ability to avoid collisions with pedestrians. Transportation for America reported that 47,700 people were killed and 688,000 others were injured due to traffic accidents in the United States from 2000-2009[4]. The APCAS was designed to drastically reduce these numbers.

The APCAS has two main objectives: safety and efficiency. In order to ensure that this system is as safe as possible all collisions should be avoided. That means the safety performance objective is to produce zero collisions. However, the system still needs to be efficient. Avoiding all possible collisions should not increase travel time; therefore, the efficiency objective is to minimize any lost time due to safety maneuvers.

1.3 Definitions, acronyms, and abbreviations

This section will give a definition of terms used throughout the document.

Term	Abbreviation	Definition
Automated Pedestrian Collision Avoidance System	APCAS	The name of the system that is being designed.
Brake by Wire	BBW	A sub-system of a vehicle that responds to deceleration requests by applying brake torque via electro mechanical actuators at all four wheels of the vehicle.
Collision Path	CP	A path of projected movement by an object based on current position and velocity. An

		intersection of two object's collision paths indicates a possible point of collision.
Pedestrian Collision Avoidance Algorithm	PCA Algorithm	The algorithm that takes in the pedestrian's position and velocity as well as the vehicle's position and velocity to determine if a collision is imminent.
Pedestrian Sensor	PDS	A sensor on the vehicle that detects a pedestrian's position and velocity then sends this information to the Safety Controller (SC).
Safety Controller	SC	Contains the PCA algorithm which gathers vehicle speed data from the vehicle in order to determine if an outgoing brake request is necessary
Steady State Velocity	SSV	A target velocity at which the vehicle will be traveling at or will be returning to.

1.4 Organization

The remainder of this document is organized as follows.

Section 2 describes the key functions that the software performs, the intended users, a list of constraints, and assumptions regarding the system. Section 3 is an enumerated list of the requirements that the ACPAS fulfills. Section 4 gives structural and behavioral models of the specific requirements for the APCAS. Section 5 gives an overview on how to access and run the prototype. Section 6 gives a list of all documents referenced in the SRS. The final section of this document is the Point of Contact. This section gives specific information on who can be contacted to obtain more information on this system.

2 Overall Description

The general factors that affect the APCAS and its requirements are covered in this section. The topics addressed are the system's perspective and context, functionality, user characteristics, constraints, assumptions about the system and any requirements determined to be potentially implemented in future system releases but not in the current project context.

2.1 Product Perspective

The APCAS is implemented as a part of an autonomous vehicle, with the purpose of automatically avoiding pedestrians; it is enabled upon putting the vehicle in drive. The system consists of three different subsystems—PDS, SC, and BBW.

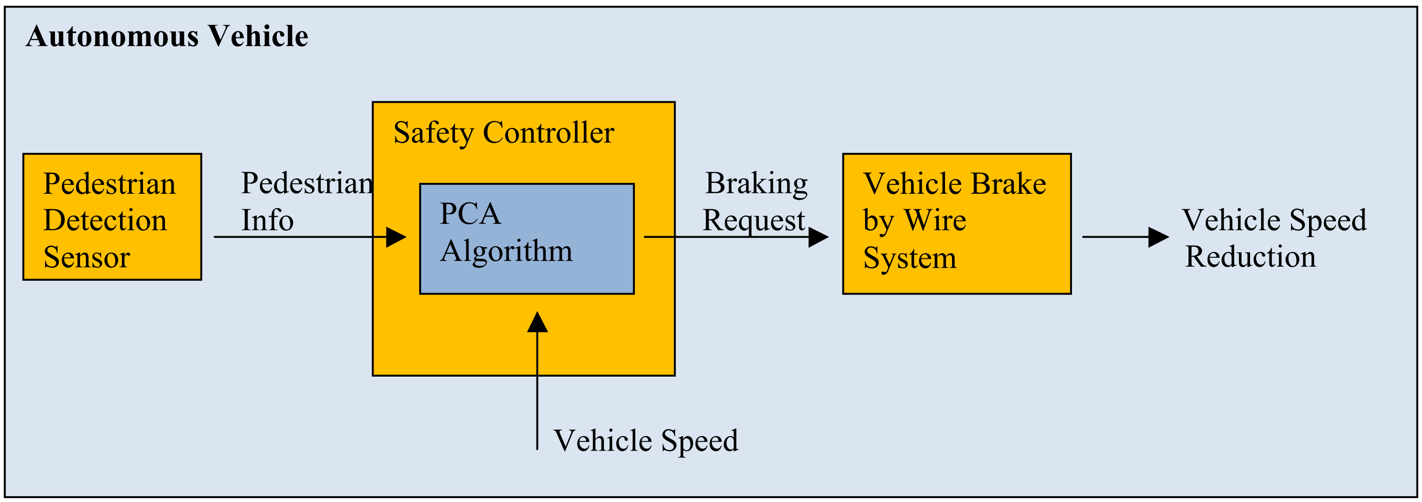


Figure 2.1.1

Figure 2.1.1 represents the three subsystems and how they interact with one another[2]. APCAS operates under several constraints, including interface, system, user, hardware, software, communication and operations constraints.

2.1.1 System Interface

System interface constraints consist of the three subsystems of the APCAS: the PDS, SC, and BBW subsystems. The PDS interacts with the SC by sending it pedestrian detection data. The SC uses this data to determine if a collision is imminent via the PCA algorithm, in which case it sends a brake apply request to the BBW. The BBW then applies brakes on all four wheels.

2.1.2 User Interface

The user interface constraints are minimal, due to low user interaction with the APCAS. The user has the option to disable/enable the APCAS via buttons on the steering wheel or an LED display provided by the vehicle's manufacturer. If the user disables the APCAS, the vehicle cannot operate autonomously (e.g. the user must manually operate the vehicle).

2.1.3 Hardware Interface

Hardware interface constraints apply to the hardware components that make up the subsystems of the APCAS.

The PDS utilizes a sensor camera that is mounted on the inside of the windshield next to the rear-view mirror in order to minimize obstruction of view for the driver. It obtains pedestrian location relative to the car with an accuracy of +/- 0.5 meters, pedestrian velocity within +/- 0.2 meters per second accounting for a direction of +/- 5 degrees, and sends a data packet to the SC every 100 milliseconds[2].

The SC consists of a PCA algorithm that calculates imminent collisions based on the data from the PDS.

Finally, the BBW decelerates the vehicle by applying brake torque via electro mechanical actuators at all four wheels of the vehicle, and sensing the actual vehicle deceleration for closed loop control. It has a deceleration accuracy of +/- 2 percent, a response time to each requested deceleration of 150 milliseconds, a release time of 200 milliseconds, and a maximum deceleration of 0.85 g[2].

2.1.4 Software Interface

The software interface constraints include the vehicle's stock user interface in order to receive waypoint and destination data input from the user. This provides the autonomous vehicle with the user's desired traveling location.

2.1.5 Communications Interface

The communications constraints contain the vehicle's use of a built-in GPS system of in order to navigate to the user's desired location.

2.1.6 Memory Constraints

There are no memory or site adaption operations constraints on the APCAS system for this current project.

2.1.7 Operations

The operation constraints involve the user interaction with enabling/disabling the APCAS via user overrides of the system. The user can override the APCAS using two methods: selecting the disable button on the user interface or steering wheel, and engaging the brake/gas pedal or manually steering the vehicle. In both cases, autonomous vehicle operation (automatic driving) becomes manually operational by the user and the APCAS is disabled.

2.2 Product Functions

The APCAS will monitor the path in front of the vehicle during autonomous vehicle operation (driving), searching for and identifying potential collisions with pedestrians. In order to perform this task, the APCAS will use the PDS, SC and BBW subsystems.

The PDS will determine a potential collision by analyzing the collision path between the vehicle and pedestrian. Data collected by the PDS will be sent as a packet every 100 milliseconds to the SC. The PCA algorithm in the SC will determine if a collision is imminent, in which case will send a brake request to the BBW. In order to decelerate and avoid the collision, the BBW will automatically interrupt the steady-state velocity control, or cruise control, of the vehicle and apply braking at all four wheels. Once the collision has been successfully avoided and the path in front of the vehicle is clear, the vehicle's velocity will automatically return to its SSV [2].

2.3 User Characteristics

Since the APCAS is automated and implemented on a fully autonomous vehicle, the expectations about the user are minimal. The user will be expected to have a valid driver's license, and obey all traffic and driving laws as required by the state they are operating the vehicle in. These expectations are to ensure the best possible safety of the user and pedestrians in case the user chooses to override the system by either disabling it through the user interface or by manually operating the vehicle via the gas and brake pedals or steering wheel.

2.4 Constraints

There are certain constraints on the APCAS that detail safety-critical and system-dependent properties.

- The APCAS will not continuing working if it suffers a hardware failure such as a non-functioning PDS (i.e. blocked lens.)

- Certain legal constraints to be aware of include when the vehicle is decelerating, brake lights on the rear of the vehicle will turn on. Also, upon disabling the APCAS, the vehicle can no longer drive autonomously.

- The PDS only has the ability to detect information of one pedestrian at a time, not multiple pedestrians.

- The only avoidance maneuver the APCAS will perform is braking. Therefore, the system will not use any steering functionality to avoid pedestrians.

2.5 Assumptions and Dependencies

Under each scenario involving the use of the APCAS, several assumptions and dependencies are made about the vehicle, pedestrian, and hardware components.

The vehicle is assumed to start at a speed of SSV, or 50 KPH. After collision avoidance, the vehicle will accelerate at a rate of .25 g until it reaches the steady state velocity. The direction that it is heading will always be straight, corresponding to the positive x-axis. The vehicle's position will start at the coordinates of (0 meters, 0 meters) on an (x, y) plane[2].

The pedestrian is assumed to start at either a constant speed, or be static; the velocity of the pedestrian can change with infinite acceleration. The direction that the pedestrian is heading will always be parallel to the y-axis, with an initial position of (35 meters, -7 meters) on an (x, y) plane[2]. The size of the pedestrian is assumed to be a circle that is half a meter in diameter.

The PDS is assumed to have a range of up to 40 meters. Additionally, it is assumed that the PDS can only detect one pedestrian at a time. The system must automatically disable upon detecting a non-functioning sensor. The SC subsystem is dependent on information received from the PDS as well as information received from the vehicle regarding vehicle speed. The BBW is dependent on information received by the SC.

2.6 Apporportioning of Requirements

For the current project, the PDS is assumed to detect one pedestrian at a time. In future releases of the APCAS, the PDS will be able to detect multiple pedestrians at a time. Future releases of the APCAS might also include adding additional avoidance maneuvers such as steering to avoid a pedestrian.

3 Specific Requirements

Based on interactions with the customer, the list below is a summary of the APCAS's requirements.

System start-up:

1. The system must be activated immediately after the vehicle is shifted into drive.

PDS:

2. The PDS will be a stereo camera that will be placed directly behind the windshield next to the rear view mirror and be facing forward.
3. If the lens is covered, then the camera must stop all readings and warn the driver.

4. An errant pedestrian signal and/or an errant brake request (hardware failure) cannot cause an unwanted, hard brake apply. A hard brake apply is determined to be greater than .25 g (1 g = 9.81 m/s^2.)

BBW:

5. Will be activated by a braking request sent by the SC.
6. While the brakes are being applied, the brake lights should be turned on in order to alert the driver behind that the vehicle is braking.

SC:

7. SC must read in the vehicle's velocity as well accept pedestrian information that was sent over by the PDS.
8. PCA Algorithm uses the pedestrian information as well as the vehicle speed to calculate if a collision will occur.
    a. If a collision is imminent, then the PCA Algorithm will calculate the rate at which the vehicle must decelerate in order to avoid the pedestrian.
    b. The SC will then send the rate that the vehicle should brake at to the BBW.
    c. Once the SC determines the collision has been avoided it will return to SSV.
9. If a collision is not imminent, then the vehicle will maintain the SSV.

Alerts:

10. An auditory alert must be issued when the pedestrian sensors detects that it is functioning incorrectly. This alert should be a sequence of high-pitch sounds.
11. There must also be a visual alert. Visual alerts will be in the form of an LED display near the windshield of the vehicle.
12. The system will be disabled immediately after the alert is sounded. If there is no driver interaction with the vehicle at this time, the vehicle will slow to a stop.

System Shutdown:

13. The system must be deactivated immediately after the vehicle is shifted out of drive.

User Interaction:

14. The system must be disabled if the driver engages the gas pedal, brake pedal, or the steering wheel.
15. The system must be re-enabled immediately after the driver presses a button located on the dashboard of the vehicle.

4 Modeling Requirements

4.1 Use Case Diagram

A use case diagram is a list of steps, typically defining interactions between a role and a system, to achieve a goal. The actor can be a human or an external system. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholders goals [5].

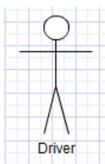


Figure 4.1.1

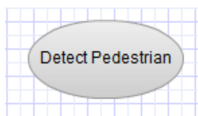


Figure 4.1.2

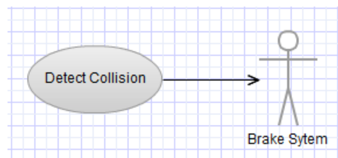


Figure 4.1.3

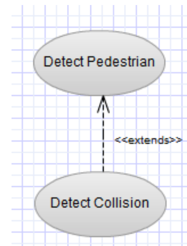


Figure 4.1.4

Figure 4.1.5

Figure 4.1.1 displays an actor that is a direct external user of a system. Figure 4.1.2 displays an oval that represents a use case that represents a specific function of the system. Figure 4.1.3 displays an arrowed line which represents an interaction between two objects. Figure 4.1.4 displays an extend relationship. An extend relationship specifies that a particular use case extends the behavior of another. Figure 4.1.5 displays a box that represents the systems boundary. The systems boundary defines the scope of the system[6].

Team Pedestrian Safety System Use Case Diagram

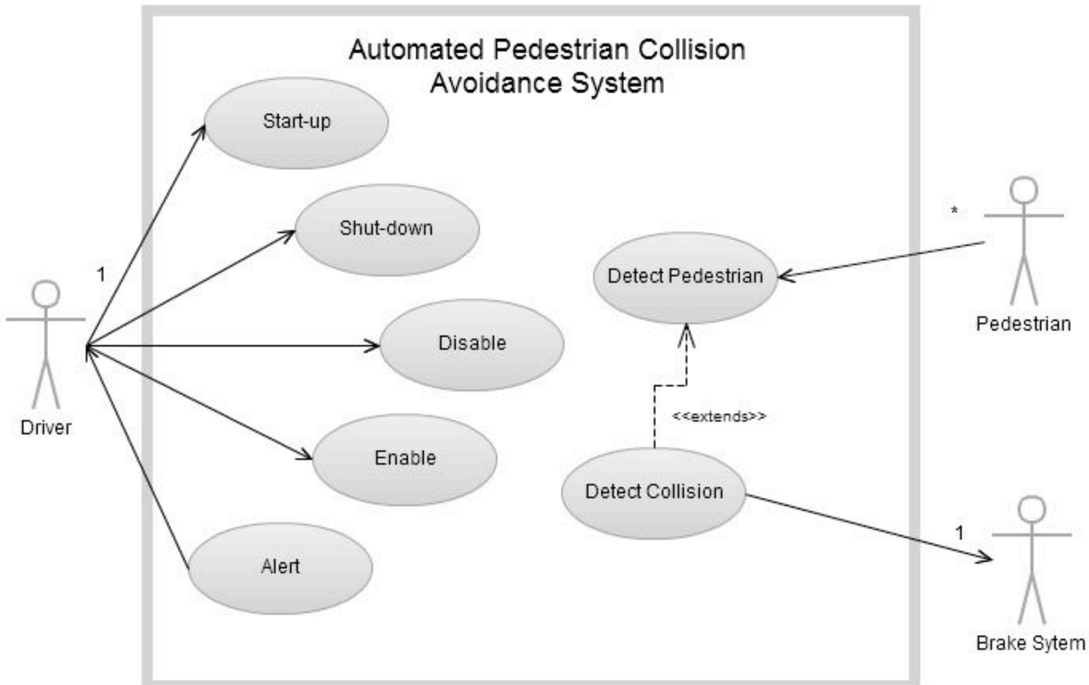


Figure 4.1.6

Table 4.1.1

Use Case:	***Start-up***
Actors:	Driver
Descriptions:	The entire system must be able to start up immediately after the driver shifts the vehicle into drive.
Type:	Primary
Cross-refs:	1
Use Cases:	All

Use Case:	***Shut-down***
Actors:	Driver

Use Case:	Shut-down
Descriptions:	The system is shut down when the driver shifts the vehicle out of drive.
Type:	Primary
Cross-refs:	13
Use Cases:	All

Use Case:	Disable
Actors:	Driver
Descriptions:	The driver can disable the system by engaging the gas pedal, brake pedal, or the steering wheel. The system will also be disabled if the PDS is not functioning correctly.
Type:	Primary
Cross-refs:	12, 14
Use Cases:	All but Start-Up and Shut-Down

Use Case:	Enable
Actors:	Driver
Descriptions:	If the system is disabled the driver can press a button which will enable the system.
Type:	Primary
Cross-refs:	15
Use Cases:	All but Start-Up and Shut-Down

Use Case:	Alert
Actors:	Driver
Descriptions:	If an error occurs within the system the driver will be alerted and the system will be disabled.

Use Case:	*Alert*
Type:	Primary
Cross-refs:	10, 11, 12
Use Cases:	None

Use Case:	*Detect Pedestrian*
Actors:	Pedestrian
Descriptions:	If the PDS detects a pedestrian. It will send the pedestrian's location and velocity to the SC.
Type:	Primary
Cross-refs:	2, 3, 4
Use Cases:	Detect Collision

Use Case:	*Detect Collision*
Actors:	Brake System
Descriptions:	If a pedestrian is detected, the SC will use the pedestrian's location and velocity, as well as the vehicle's speed to determine if a collision is possible. If a collision is possible, then the SC will determine the necessary rate of deceleration to avoid the collision and send it to the BBW.
Type:	Primary
Extends:	Detect Pedestrian
Cross-refs:	5, 6, 7, 8, 9
Use Cases:	None

4.2 Class Diagram

A class diagram depicts the static structure of a system by showing the classes, attributes, operations, and relationships of the different classes within the system.

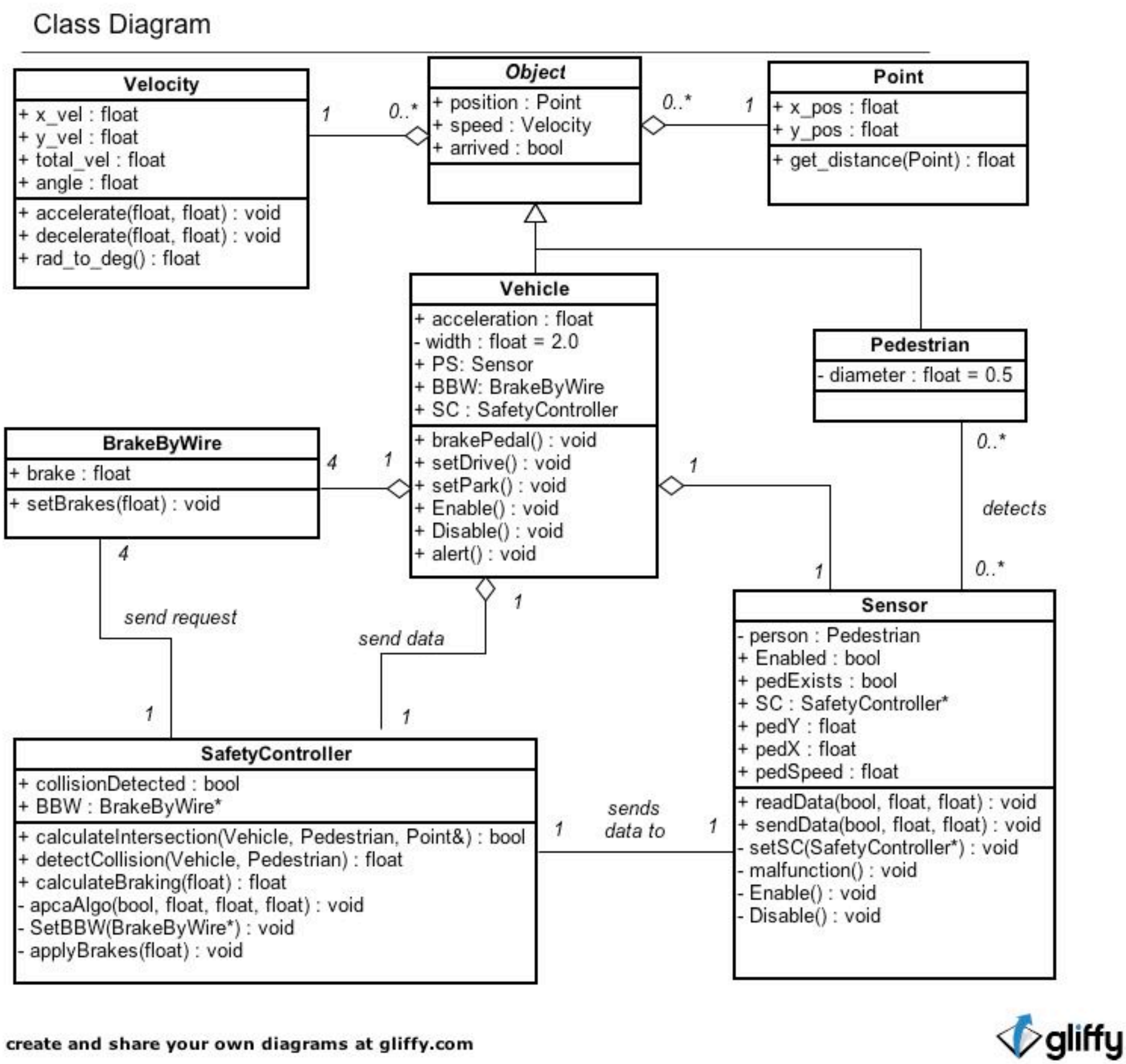


Figure 4.5.1: Class Diagram

Within Figure 4.5.1, some of the different types of relationships used include association, aggregation, and inheritance. Each association has a name and is used to show how two classes interact with one another. Aggregation is more specific than association and can be thought of as a “has a” relationship. Each aggregation is represented by white diamond on one end of the association. Inheritance can be thought of as an "is a" relationship among two classes and is represented by a white triangle at the end of an association. The triangle points to a class called the parent class. In Figure 4.5.1, the parent class is an abstract class, which it represented by italicizing the name of the class. This means that the sub class will inherit all of the attributes and operations used in the parent class. In this case, the Vehicle and Pedestrian Class will have the attributes and operations of the *Object* class.

Table 4.5.1: Data Dictionary

Element Name		**Description**
BrakeByWire		This class is responsible for sending messages to a vehicle in order to reduce its velocity.
Attributes		
	brake : float	The amount the vehicle will brake by
Operations		
	setBrakes(amount : float) : void	Reduce the speed of a vehicle by the amount specified.
Relationships	BrakeByWire is a part of the Vehicle class. Every vehicle will have four brakes that reduce the vehicles velocity when it receives a request from the SafetyController.	
UML Extensions		

Element Name		**Description**
Object		This is an abstract class that defines a moving actor within the system.
Attributes		
	position : Point	The position of the object
	speed : Velocity	The velocity of the object
	arrived : bool	Indicates if the object has arrived in the collision area
Operations		
Relationships	This class acts as the parent class from the Vehicle and Pedestrian class	
UML Extensions		

Element Name	**Description**
Pedestrian	This class defines a pedestrian that would be walking down the street, or

		standing still.
Attributes		
	diameter : float	The size of the pedestrian
Operations		
Relationships	This class will be detected by the sensor on a vehicle and is inherited from the *Object* class.	
UML Extensions	Tagged values: diameter = 0.5 meters	

Element Name		**Description**
Point		A data type to mimic the features of a point in space.
Attributes		
	x_pos : float	The X coordinate in space
	y_pos : float	The Y coordinate in space
Operations		
	get_distance(p : Point) : float	Takes in a Point to find the distance between. Returns a float with the distance between the two points.
Relationships	This class is a part of the *Object* class.	
UML Extensions		

Element Name		**Description**
SafetyController		This class acts as a mediator between a Vehicle, a Sensor, and a BrakeByWire. It processes information and determines if the car needs to slow down because of a collision.
Attributes		
	BBW : BrakeByWire*	A pointer that points to the BBW that it will send requests to
	collisionDetected : bool	True if a collision is detected between a Vehicle and Pedestrian. Otherwise, false.
Operations		
	SetBBW(BrakeByWire*) : void	Sets the BBW that it will send requests to.
	apcaAlgo(pedExists : bool, pedX : float, pedY : float, pedSpeed : float) : void	Calls the algorithm to start processing a pedestrians information if a pedestrian exists.
	calculateIntersection(car : Vehicle, person : Pedestrian, intersection :	Calculates a possible point of intersection between a Pedestrian and a Vehicle. The intersection

	Point&) : bool	point is passed through by reference. Returns true if there is a point of intersection. Otherwise, false.
	detectCollision(car : Vehicle, person : Pedestrian) : float	Detects if a collision is possible between a Vehicle and Pedestrian. Returns the distance the car is away from the intersection. Returns -1 if no collision is detected.
	applyBrakes(brake : float) : void	Sends the amount to brake to the brake by wire system.
	calculateBraking(dist : float) : float	Calculates the amount needed to brake using the distance the car is from the intersection point. Returns the amount the car must decelerate to avoid a collision.
Relationships	This class will receive data from the Vehicle and Sensor class. It will also send requests to the BrakeByWire class in order to reduce the velocity of a vehicle that it is a part of.	
UML Extensions		

Element Name		**Description**
Sensor		This class will detect a pedestrian that is walking toward the path of the vehicle that the camera is mounted on
Attributes		
	pedExists : bool	True if a pedestrian has been detected by the sensor. False otherwise.
	SC : SafetyController*	A pointer to the safety controller that the sensor will be sending information to.
	Enabled : bool	True if the sensor is enabled. False otherwise.
	pedX : float	The X coordinate of the pedestrian's position
	pedY : float	The Y coordinate of the pedestrian's position
	pedSpeed : float	The speed of the pedestrian
	person : Pedestrian	A pedestrian that the sensor detects
Operations		
	malfunction() : void	Determines if the sensor is not working properly. Notifies the vehicles the sensor is a part of.

	setSC(SafetyController*) : void	Sets the safety controller that the sensor will be sending information to.
	readData(pedExists : bool, pedY : float, pedSpeed : float) : void	Reads data about a pedestrian and stores it.
	Enabled() : void	Enables the sensor to start reading in data
	Disable() : void	Disables the sensor and stops reading in data
	sendData(pedExists : bool, pedY : float, pedSpeed : float) : void	Forwards the data it read from the pedestrian to a safety controller
Relationships	This class is a part of the Vehicle class in the sense that every car has a sensor. It also detects pedestrians in order to send the information along to the safety controller	
UML Extensions		

Element Name		**Description**
Vehicle		This class defines a vehicle that would be driving down the street.
Attributes		
	PS : Sensor	The sensor that is a part of the vehicle.
	BBW : BrakeByWire	The brakes that are a part of the vehicle.
	SC : SafetyController	The safety controller that is a part of the vehicle.
	acceleration : float	The acceleration of the vehicle. Value is negative if the car is decelerating.
	width : float	The width of the vehicle.
Operations		
	setDrive() : void	Sets the car in drive. Upon this, the APCAS is enabled.
	setPark() : void	Sets the car in park. Upon this, the APCAS is disabled.
	Enable() : void	Turns the APCAS on.
	Disable() : void	Shuts off the APCAS.
	alert() : void	Sends a message to the driver alerting him or her that the APCAS has been disabled.
	brakePedal() : void	Turns off the autonomous car and disables the APCAS if the driver has hit the brake pedal.

Relationships	The Vehicle class is inherited from the *Object* class. Each vehicle has four BrakeByWire brakes, a SafetyController object, and a Sensor camera. The SafetyController will be able to receive data from the Vehicle.
UML Extensions	Tagged values: width = 2.0 meters.

Element Name		**Description**
Velocity		A data type to mimic the speed and direction of an object
Attributes		
	x_vel : float	The X value of the total velocity
	y_vel : float	The Y value of the total velocity
	total_vel : float	The velocity of the object
	angle : float	The angle at which the object is traveling
Operations		
	accelerate(amount : float, interval : float) : void	Increases the velocity to accelerate the object by the amount specified
	decelerate(amount : float, interval : float) : void	Decreases the velocity to decelerate the object by the amount specified
	rad_to_deg(angle : float) : float	Converts the angle the object is traveling from radians to degrees
Relationships	This class is a part of the *Object* class	
UML Extensions		

Shown above in Table 4.5.1 is a data dictionary for the class diagram in Figure 4.5.1. This data dictionary defines each term in Figure 4.5.1 and provides explicit clarification for any system-specific uses of broad terms. It explains interactions, and it also uses reader-friendly cross-references to other entries for easy understanding.

4.3 Sequence Diagrams

A sequence diagram shows the participants in an interaction and the sequence of messages among them. A sequence diagram shows the interaction of a system with its actors to perform all or part of a use case.

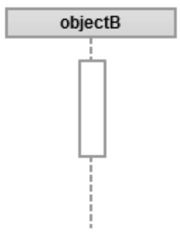


Figure 4.3.1

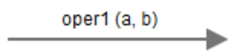


Figure 4.3.2

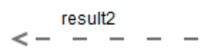


Figure 4.3.3

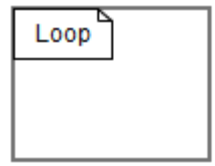


Figure 4.3.4

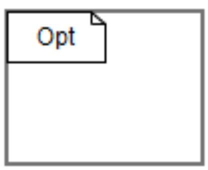


Figure 4.3.5

Figure 4.3.1 displays a representation of an instantiated object and its lifeline. Figure 4.3.2 displays an outgoing message from one object to another. Figure 4.3.3 displays a return message. Messages show all of the interactions between the objects. Figure 4.3.4 displays a loop. Items inside the loop repeat while the loops condition is satisfied. Figure 4.3.5 displays an option. The items inside the option occur only if its condition is satisfied[6].

4.3.1 Scenario 1

1. User shifts the vehicle into drive.
2. PDS is enabled.
3. PDS collects data from the pedestrian.
4. PDS sends data about the pedestrian to the SC.
5. SC processes the data.
6. If a collision is possible the SC sends an amount to decelerate by to the BBW.
7. The BBW decelerates the vehicle by that value.
8. 100ms after the PDS collects data the PDS collects new data.
9. This repeats while the system is enabled.
10. The user shifts the vehicle out of drive.
11. The PDS is disabled and the system shuts down.

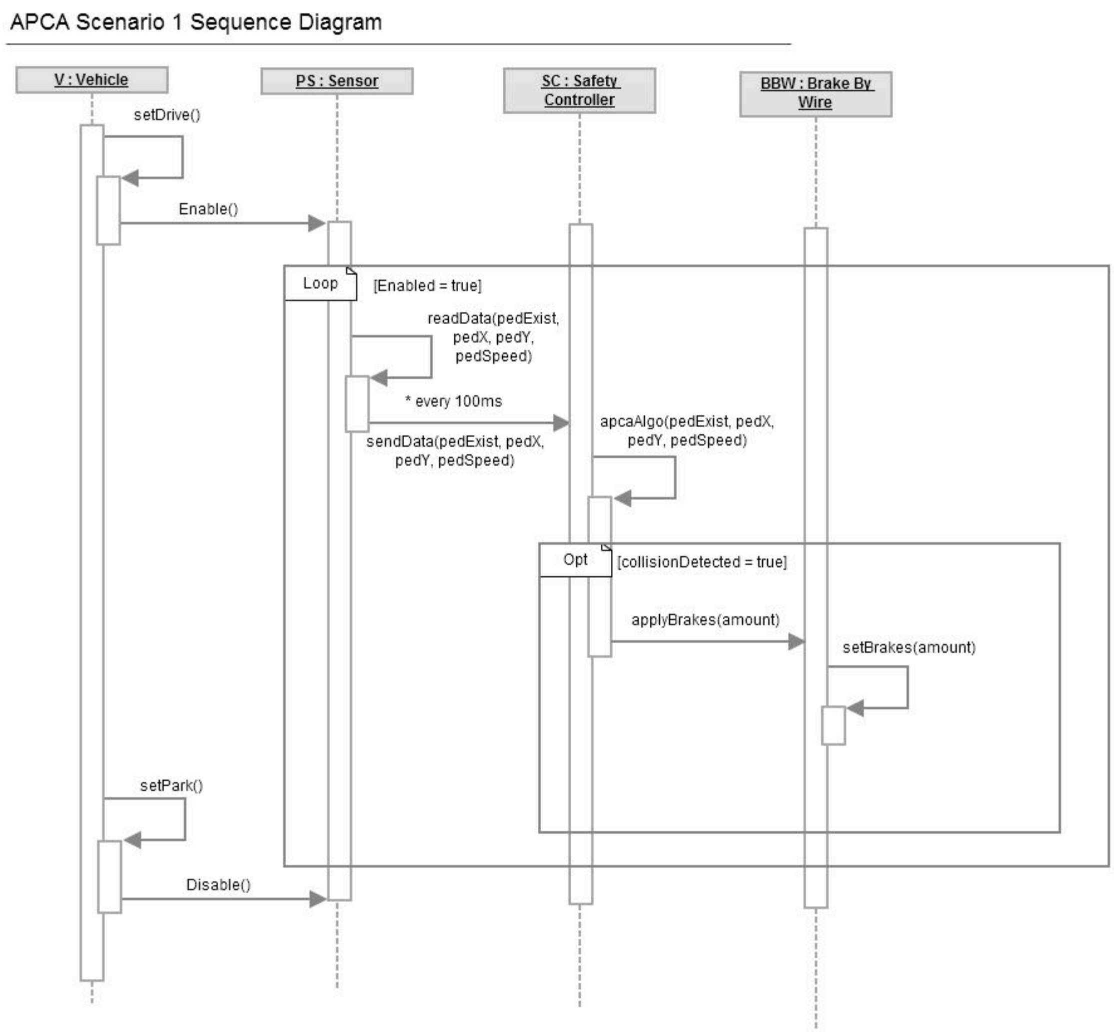
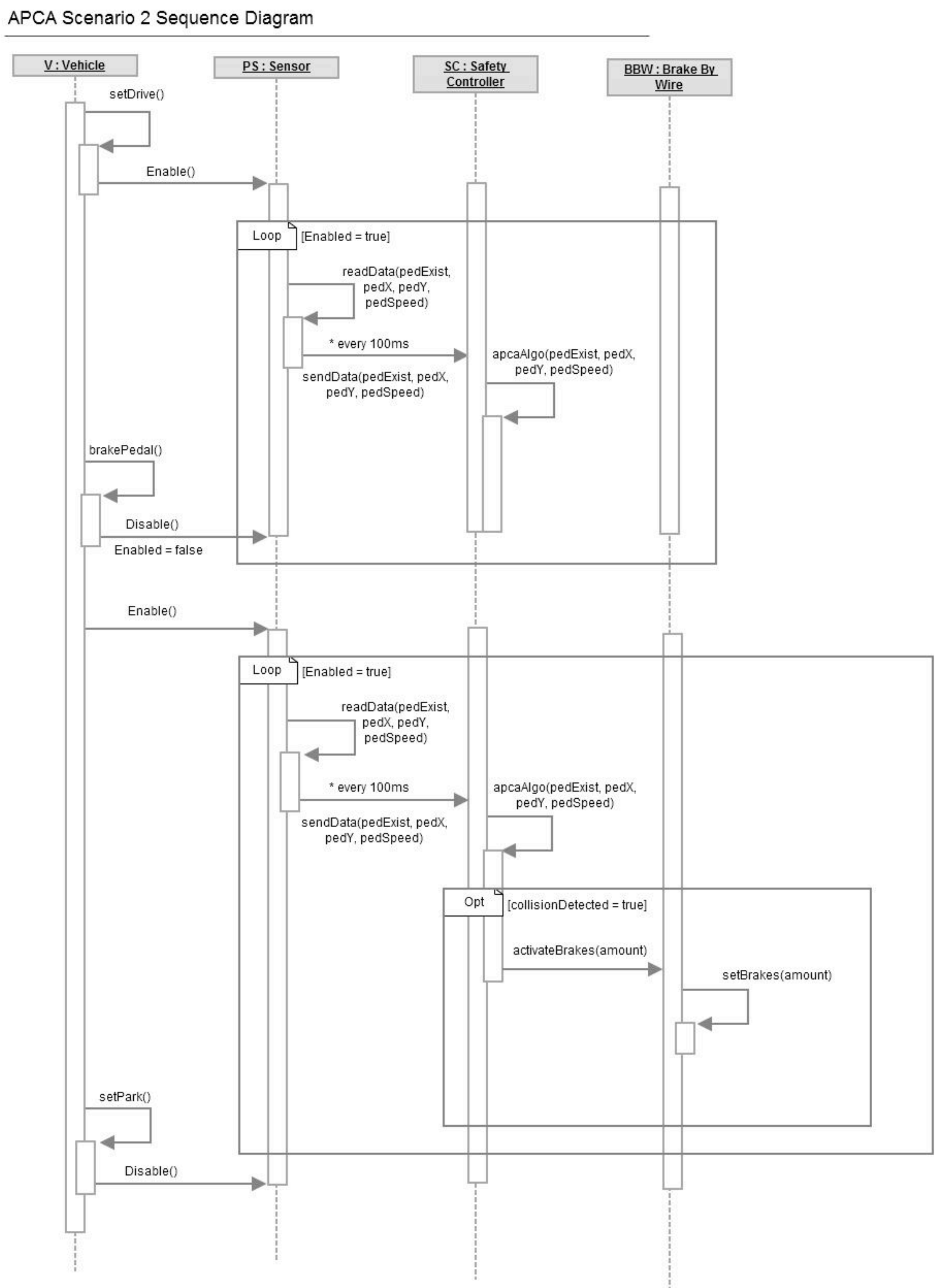


Figure 4.3.6

4.3.2 Scenario 2

1. User shifts the vehicle into drive.
2. PDS is enabled.
3. PDS collects data from the pedestrian.
4. PDS sends data about the pedestrian to the SC.
5. SC starts to process the data.
6. User presses the brake pedal.
7. System is disabled.
8. User enables the system.
9. PDS is enabled.
10. PDS collects data from the pedestrian.
11. PDS sends data about the pedestrian to the SC.
12. SC processes the data.

13. If a collision is possible the SC sends an amount to decelerate by to the BBW.
14. The BBW decelerates the vehicle by that value.
15. 100ms after the PDS collects data the PDS collects new data.
16. This repeats while the system is enabled.
17. The user shifts the vehicle out of drive.
18. The PDS is disabled and the system shuts down.



APCA Scenario 2 Sequence Diagram
V : Vehicle
PS : Sensor
SC : Safety Controller
BBW : Brake By Wire
setDrive()
Enable()
Loop
[Enabled = true]
readData(pedExist, pedX, pedY, pedSpeed)
* every 100ms
sendData(pedExist, pedX, pedY, pedSpeed)
apcaAlgo(pedExist, pedX, pedY, pedSpeed)
brakePedal()
Disable()
Enabled = false
Enable()
Loop
[Enabled = true]
readData(pedExist, pedX, pedY, pedSpeed)
* every 100ms
sendData(pedExist, pedX, pedY, pedSpeed)
apcaAlgo(pedExist, pedX, pedY, pedSpeed)
Opt
[collisionDetected = true]
activateBrakes(amount)
setBrakes(amount)
setPark()
Disable()

Figure 4.3.7

4.3.3 Scenario 3

1. User shifts the vehicle into drive.
2. PDS is enabled.
3. PDS malfunctions.
4. The user is alerted that the system has malfunctioned.
5. The PDS is disabled and the system shuts down.

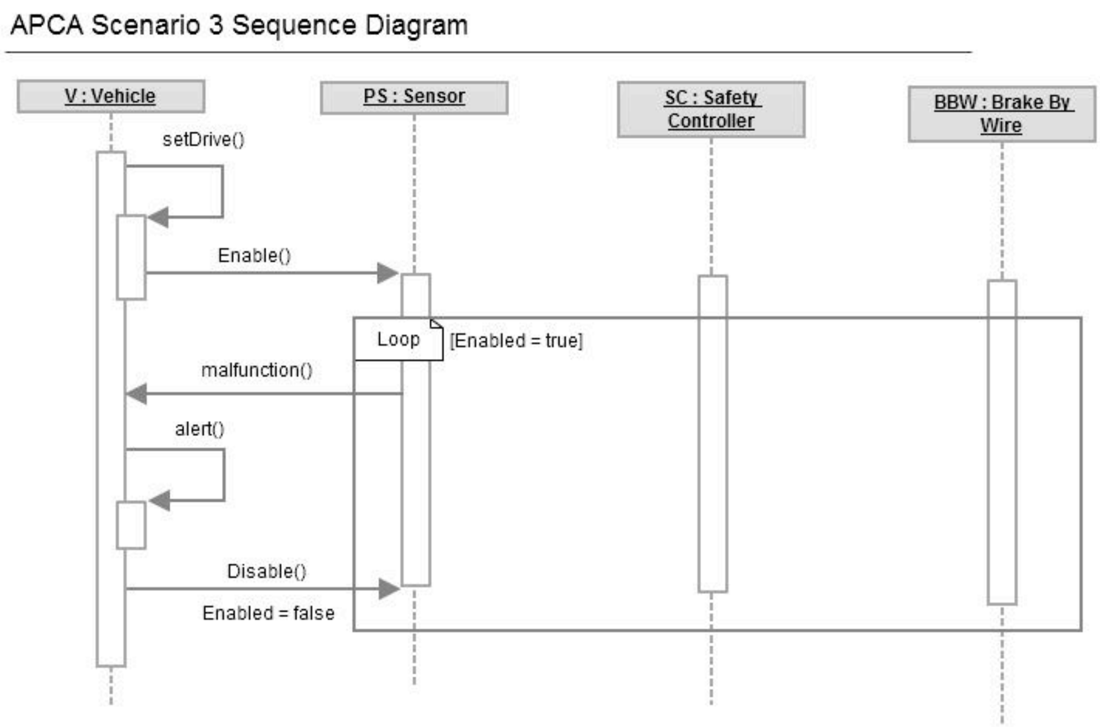


Figure 4.3.8

4.4 State Diagrams

A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states. A state diagram specifies the state sequences caused by event sequences.

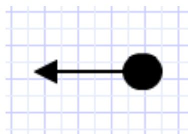


Figure 4.4.1

Figure 4.4.2

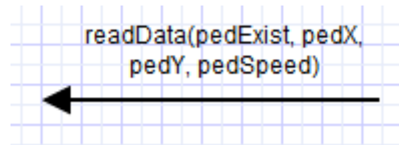


Figure 4.4.3

Figure 4.4.1 displays a solid circle with an arrow. The arrow indicates the start state. Figure 4.4.2 displays a labeled rectangle that represents an objects state. Figure 4.4.3 displays an arrowed line with an operation. These represent events that transfers an object between states[6].

4.4.1 PDS State Diagram

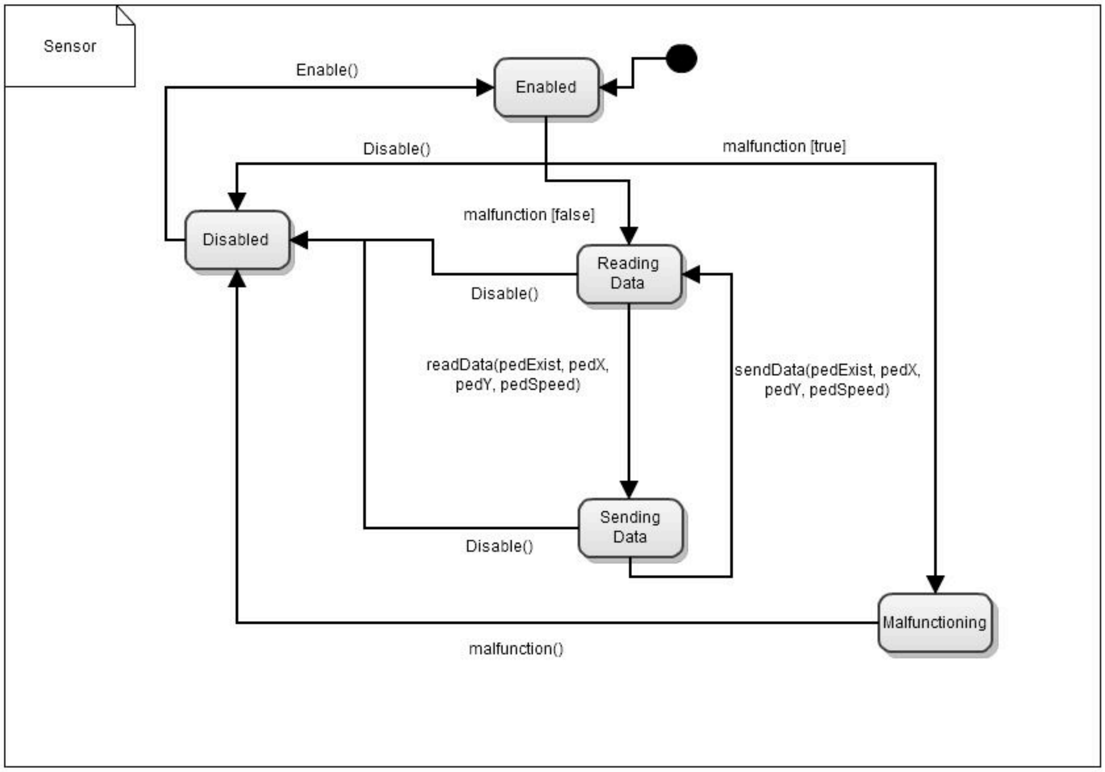


Figure 4.4.4

Figure 4.4.4 shows that the PDS is enabled by the vehicle. If it is not malfunctioning it begins reading data. After reading data for the first time the sensor sends the data to the SC. After the data has been sent, the sensor begins to read data again. 100ms after the last time the sensor sent data, the sensor sends data to the SC. This loop repeats while the system is enabled. If the sensor is malfunctioning it tells the vehicle that it is malfunctioning and the vehicle disables APCAS. The driver can disable the sensor at any time. If the sensor is disabled, the driver can enable it.

4.4.2 SC State Diagram

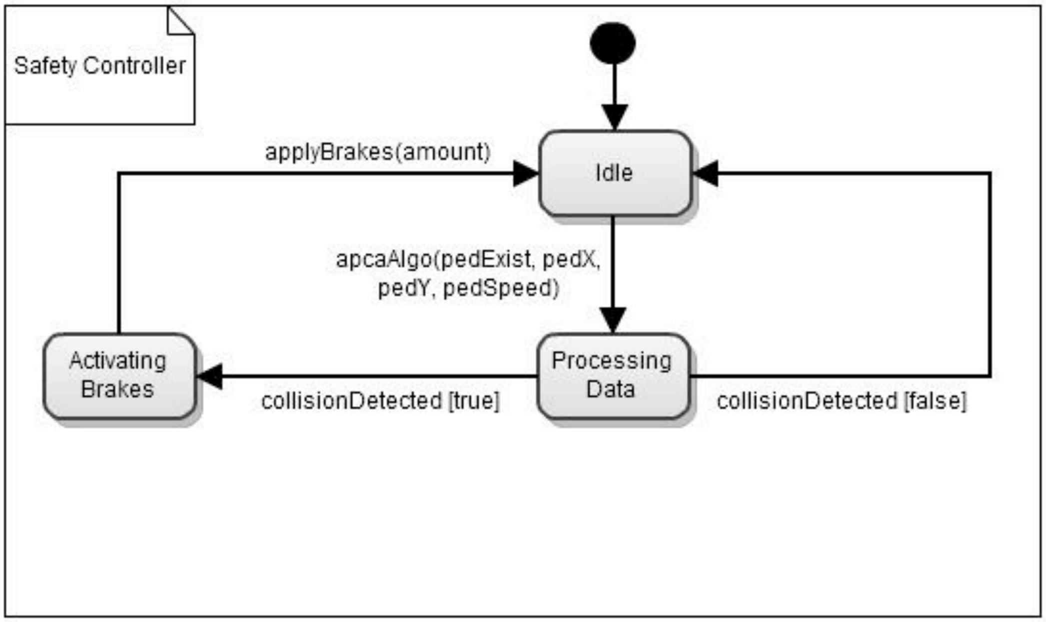


Figure 4.4.5

Figure 4.4.5 displays a SC that is idle until it is sent data from a PDS. The SC processes this data to determine if there could be a collision. If a collision is detected the SC tells the BBW object the rate at which to decelerate. Once activating the brakes has been completed the SC enters the idle state. If a collision is not detected, the SC enters the idle state.

4.4.3 BBW State Diagram

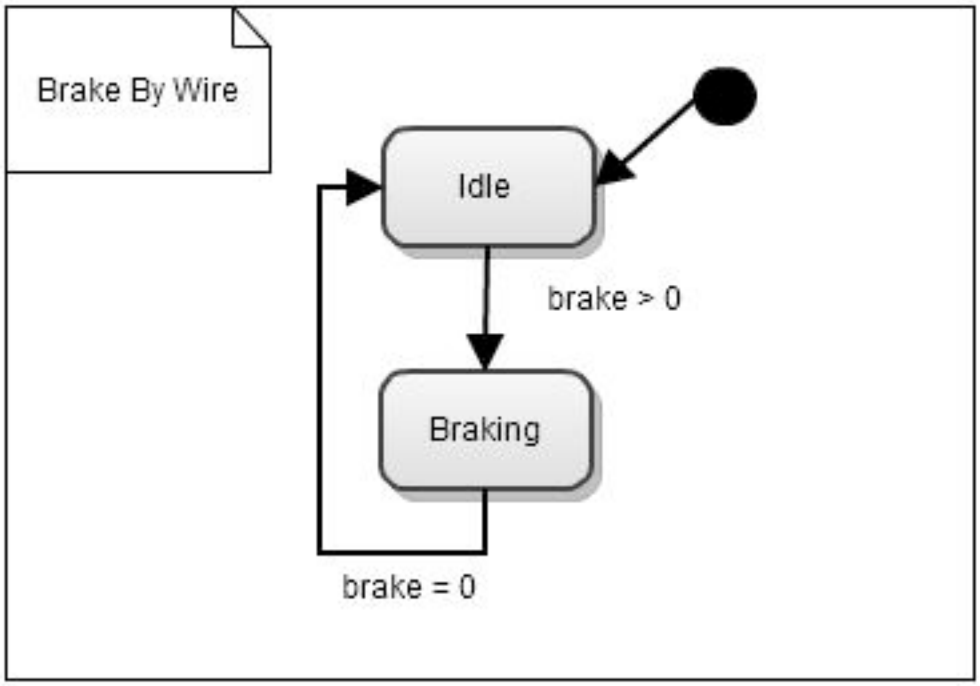


Figure 4.4.6

Figure 4.4.6 shows the BBW object is idle until the SC detects a collision. The BBW decelerates the vehicle by the rate given to it by the SC. Once braking is completed the BBW enters the idle state.

5 Prototype

Prototype v2 will simulate scenarios generated by the user, who will be able to enter various parameters for the vehicle and pedestrian or select preset scenarios. These parameters include position and velocity of the vehicle and pedestrian, end position of the pedestrian, and a delay before moving for the pedestrian. Upon the user entering these values, the scenario will be executed and the system will determine potential collisions by analyzing a collision path between the vehicle and pedestrian. Action will be taken to avoid a possible collision with the pedestrian by executing velocity reduction commands (automatic braking), which will override the current SSV of the vehicle. The braking command will activate the BBW system in the vehicle to reduce velocity as requested by the system. When the command is ended (hazard no longer exists), the vehicle’s velocity will automatically return to the SSV. Throughout the scenario, a table and graph will be constantly updated with data about the vehicle and pedestrian. Upon the scenario ending, i.e. the car reaching the end of the road or the user pressing the reset button, the graph and table will stop updating and the user will be able to get graphical representation of the movement taken by the vehicle and pedestrian.

5.1 How to Run Prototype

In order to run the APCAS prototype v2 the user must have a web browser installed on their computer and an Internet connection. On top of having a working browser, the user must have the latest flash plugin (v11.5 or higher) installed in order to run the prototype. One constraint of the prototype is that it cannot be run on any mobile iOS operating systems. These operating systems do not support flash and therefore, cannot run the prototype. Prototype v2 can be accessed through this URL, *http://www.cse.msu.edu/~cse435/Projects/F2012/APCAS-1/web/prototype.html* [3].

5.2 Sample Scenarios

Table 5.2.1: Sample scenarios

Moving then stopped				
Scenario #	Initial position, Yi	End position, Yf	Initial speed	Final speed
	(m)	(m)	(kph)	(kph)
1	-7	0	10	0
2	-7	-2	10	0
3	-7	-3	10	0
4	-7	-5	10	0
Static then moving				
Scenario #	Initial position, Yi	Delay before moving	Initial speed	Final speed
	(m)	(s)	(kph)	(kph)
5	0	1.5	0	10
6	-2	1.8	0	10
7	-4	1.1	0	10
Static				
Scenario #	Initial position, Yi	End position, Yf	Initial speed	Final speed
	(m)	(m)	(kph)	(kph)
8	0	N/A	0	0
9	-2	N/A	0	0
10	-4	N/A	0	0

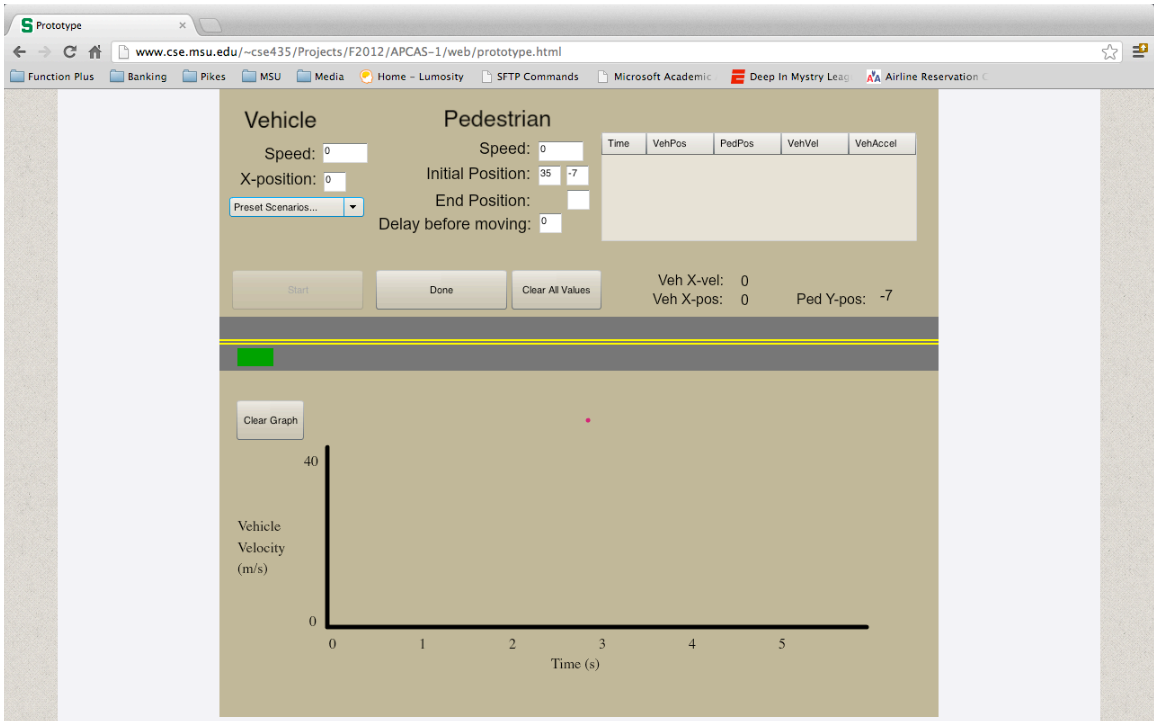


Figure 5.2.1

In order to demonstrate the functionality of the prototype, Table 5.2.1 describes a number of preset scenarios that can be selected by a drop down menu when entering the parameters into the prototype, as shown in Figure 5.2.1. Along with these ten preset scenarios, the user is able to change any of the values of the parameters in order to recreate a scenario of choice. For each scenario that is executed, the prototype will show the movement of the vehicle and pedestrian with respect to time. All scenarios that are executed will end in a non-collision state.

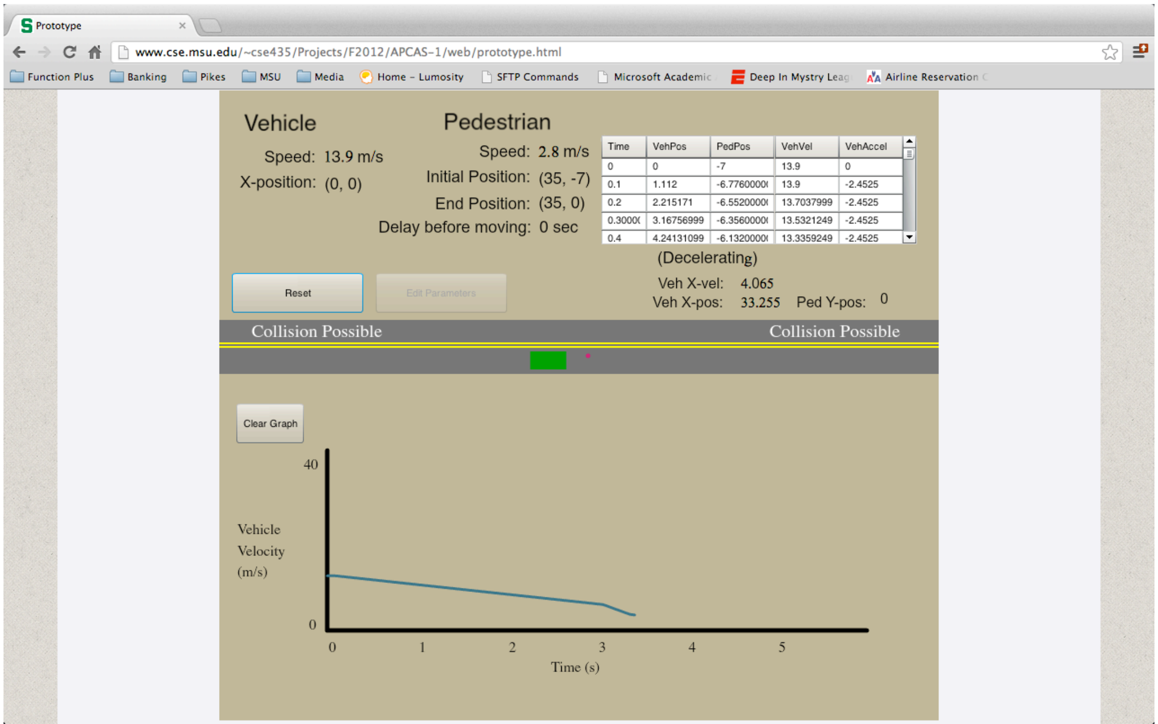


Figure 5.2.2

Scenarios one through four of Table 5.2.1 are categorized as “Moving then stopped”. This means that the pedestrian has a set end position and will cease to move once it has reached the position specified. Figure 5.2.2 shows that an end position for the pedestrian has been given and the pedestrian has stopped at that position, resulting in the car having to come to a complete stop to avoid a collision.

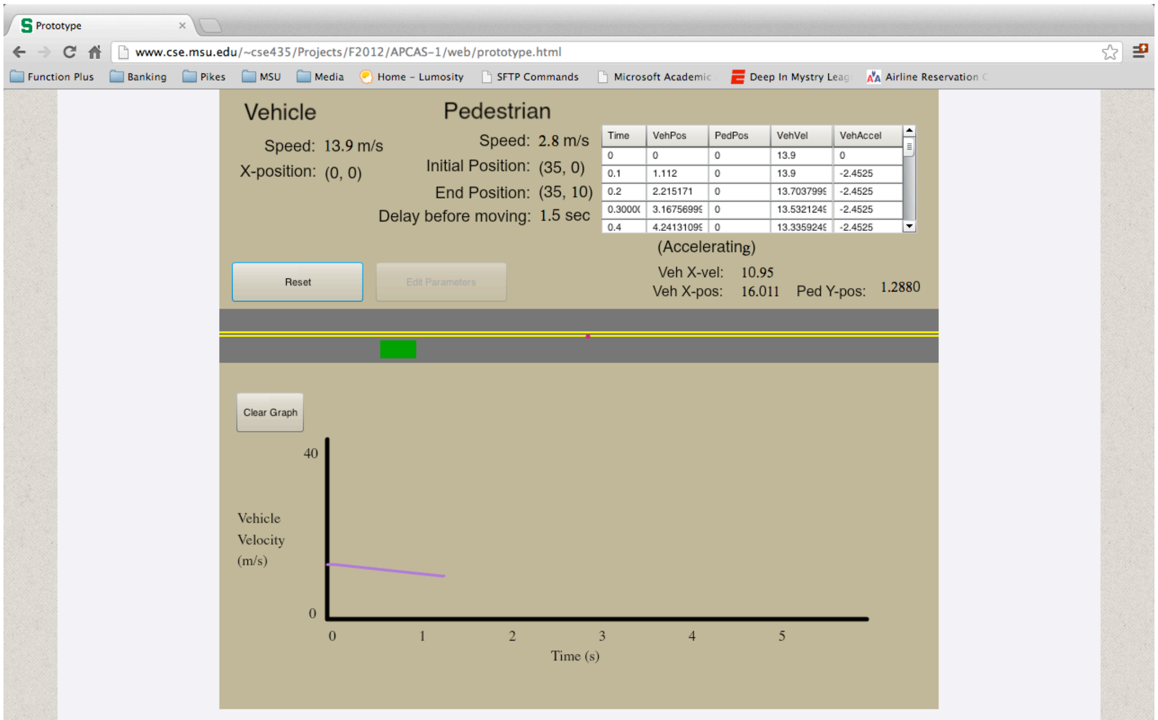


Figure 5.2.3

Scenarios five through seven of Table 5.2.1 are categorized as “Static then moving”, which means that the pedestrian will wait a certain amount of time before beginning to move. This delay time is specified in the parameters before the scenario is executed.

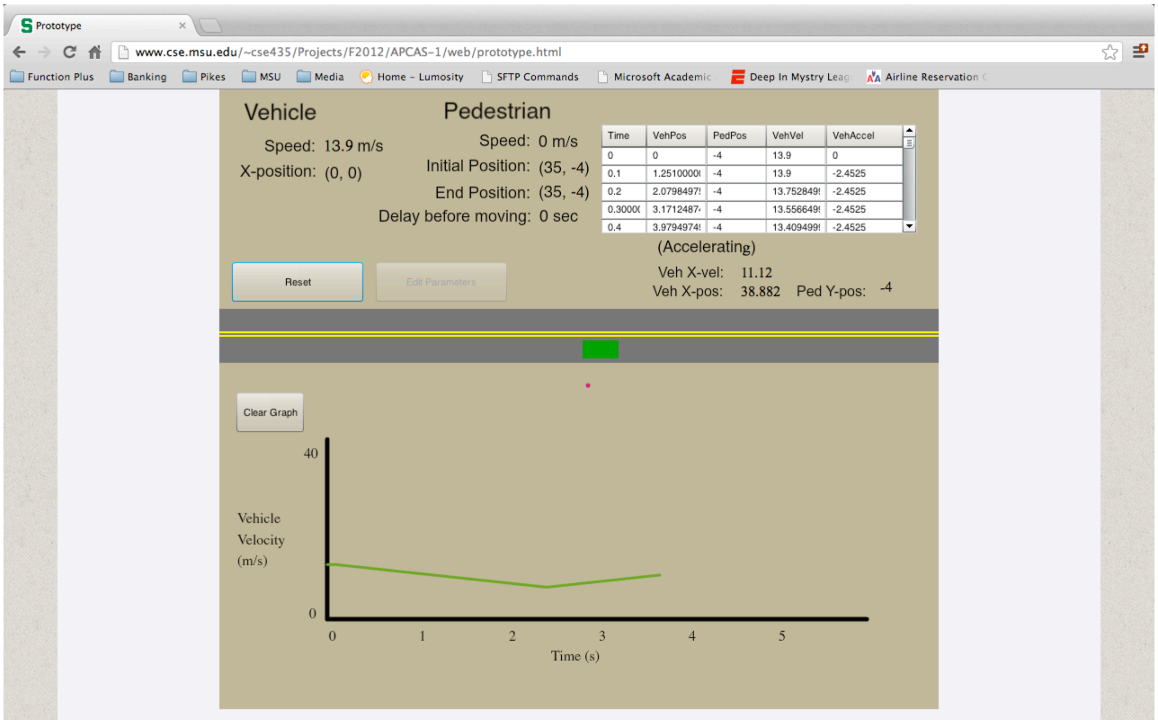


Figure 5.2.4

Scenarios eight through ten of Table 5.2.1 are categorized as “Static”. This means that the pedestrian has no initial speed and no delay before moving. It will be motionless at the specified starting point. Figure 5.2.4 shows that the pedestrian has not moved at all and the vehicle has moved passed the collision point, thereby avoiding the collision.

6 References

[1] D. Thakore and S. Biswas, “Routing with Persistent Link Modeling in Intermittently Connected Wireless Networks,” Proceedings of IEEE Military Communication, Atlantic City, October 2005.

[2] Agnew, David, Mr. "Functional Algorithm for Automated Pedestrian Collision Avoidance System." *Continental Automotive Systems, Advanced Engineering*. N.p., 23 Oct. 2012. Web. <http://www.cse.msu.edu/~cse435/Projects/F2012/Descriptions/APCA.pdf>.

[3] Project Website: Tarnowsky, Matthew, Ryan Burr, David Culham, and Bobak Shahidehpour. "APCAS-1: Team Pedestrian Safety System." *CSE 435*. N.p., 31 Oct. 2012. Web. 19 Nov. 2012. <http://www.cse.msu.edu/~cse435/Projects/F2012/APCAS-1/web/>.

[4] "Transportation For America: Dangerous by Design 2011." *Universal Feedburner*. N.p., n.d. Web. 19 Nov. 2012.

<http://t4america.org/resources/dangerousbydesign2011/>.

[5] "Use Case." *Wikipedia*. Wikimedia Foundation, 30 Oct. 2012. Web. 19 Nov. 2012. <http://en.wikipedia.org/wiki/Use_case>.

[6] Blaha, Michael, and James Rumbaugh. *Object-oriented Modeling and Design with UML*. Upper Saddle River, NJ: Pearson Education, 2005. Print.

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Betty H.C. Cheng** at Michigan State University (chengb at cse.msu.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.