Software Requirements Specification (SRS)

PCAS1 - Pedestrian Collision Avoidance System

Authors: Daniel Tan, Heather Noonan, Parker Goodrich, Ella Chen, Chris Lu

Customer: Chris Capaldi, Dataspeed Inc

Instructor: Dr. Betty H.C. Cheng, Michigan State University

1 Introduction

This document specifies the requirements for the Pedestrian Collision Avoidance System. The following sections include an introduction to the software, an overall description of the software, an in-depth look at the requirements that the software must meet, visual models of the requirements, an explanation of the prototype used to model the requirements, outside references, and a point of contact explanation. In general, this document will go over the details of what the software aims to accomplish, how the software performs its task, and what the final product will look like.

1.1 Purpose

The purpose of this document is to identify and outline the requirements of the algorithm to be created for the Pedestrian Collision Avoidance System (PCA) System. In addition, this document will explain the full details of what will be expected of the final algorithm and is intended to be reference material for the customer and instructor.

1.2 Scope

The software product to be created is an embedded automotive system named the Pedestrian Collision Avoidance System (PCA) System. The goal of the PCA System is to avoid pedestrian collision by using data from a vehicle-mounted camera sensor to detect pedestrians and engage the brake-by-wire system. In addition to preventing collisions, the algorithm must be able to keep lost time to a minimum. Security measures are also needed to prevent any hacking or system attacks.

1.3 Definitions, Acronyms, and Abbreviations

This list displays the important vocabulary used in this document, along with their definitions to allow for complete understanding in the document.

1.3.1 PCA: Pedestrian Collision Avoidance

1.3.2 Brake-by-Wire Actuator: The system component responsible for engaging a vehicle's brakes.

1.3.3 Stereo Camera (Pedestrian Detection Sensor): System component mounted on a vehicle that looks ahead and detects any pedestrians in the vehicle's path.

1.3.4 Safety Controller: The system component that uses the collision avoidance algorithm to make a decision on braking for an imminent collision.

1.3.5 Lost time: Time difference (in seconds) between system on and system off to reach a common point beyond the pedestrian with controlled vehicle back again at steady state velocity.

1.4 Organization

The remainder of this document is organized as follows. Section 2 includes an overall description of the PCA algorithm. Section 3 discusses all of the requirements for the PCAS. Section 4 shows the system in different models, as well as the prototype. Section 5 acknowledges the different references used for information to complete this document. Section 6 gives the point of contact for any extra information regarding this document.

2 Overall Description

This section of the document is responsible for giving an overview of where the product fits into the overall vehicle's system, detailing the product's major functions, as well as the constraints and assumptions taken into consideration when developing the product. Finally, this section elaborates on some ideas for building upon this product in the future.

2.1 Product Perspective

The PCA System is part of the vehicle's larger Safety Controller System. Its functionality relies on 2 hardware interfaces: the Pedestrian Detection Sensor and the Brake-by-Wire Actuator.

The Pedestrian Detection Sensor is a stereo camera that is responsible for pedestrian recognition and tracking. It outputs the pedestrian's (x, y) coordinates relative to the vehicle (with accuracy +/-.5 m), and pedestrian's velocity (with +/- .2m/s speed accuracy and +/- 5 deg direction accuracy). It then sends this information in packets to the PCA System every 100 ms.

The Brake-by-Wire Actuator is responsible for responding to deceleration requests by interrupting the steady state velocity control and applying the brake torque via electro-mechanical actuators at all four wheels of the vehicle. It has a deceleration accuracy of +/- 2% with a response time of 200ms and a release time of 100 ms. The maximum deceleration that the Brake-by-Wire Actuator can provide is 0.7g.

The below diagram (Figure 1) acts as an illustration of where the PCA System fits in relation to the vehicle's other interfaces. The larger yellow boxes represent each core part of our system. The blue box represents the PCA algorithm itself, which is a part of the Safety Controller.

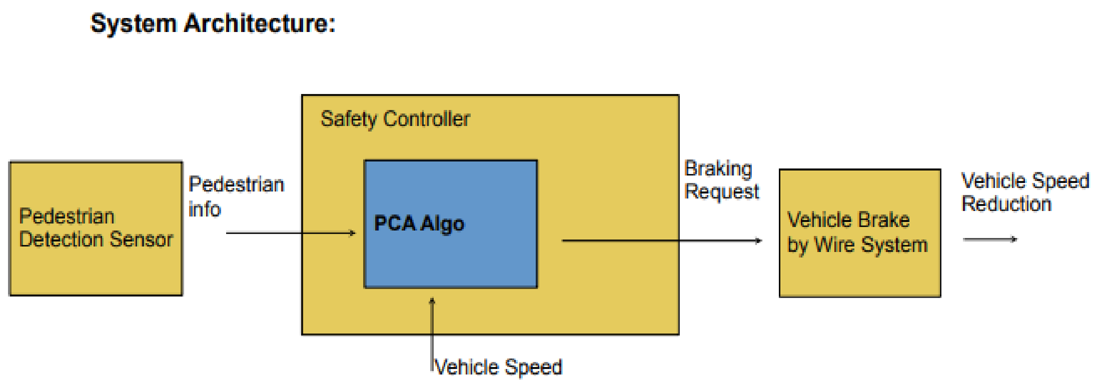


Fig.1 Data Flow Diagram. [2]

2.2 Product Functions

The major functions that the PCA software will perform include the following:

- Identifying potential collisions with pedestrians by analyzing the collision path between the vehicle and a detected pedestrian.
- Managing vehicle velocity in response to pedestrians via velocity reduction commands (automatic braking) which override the current steady state velocity of the vehicle and returning the vehicle to steady state velocity when a collision is no longer imminent. The braking command will activate the brake-by-wire system in the vehicle to reduce velocity as requested by the system.
- Identifying and mitigating opportunities in which a security breach of the system endangers a pedestrian.

2.3 User Characteristics

The user is expected to be at least 16, have a valid driver's license, and possess basic driving abilities and knowledge of car feature interfacing, such as working the radio, A/C, etc. It is also expected that the user is aware of the vehicle's PCA system and has an understanding of the various alert messages the system may send.

2.4 Constraints

The constraints for this product have been divided into two categories: Safety Critical and Performance. Properties listed in the Safety Critical section, if not met, could result in loss of life, property, or cause other significant damage. The properties listed in the Performance section must be satisfied if the system is to perform as intended.

Safety Critical

1. There shall be zero collisions between the vehicle and pedestrians in all 10 testing scenarios described in the project proposal.
2. There should be no outside access to the algorithm while the car is in motion.
3. Security breaches should be minimized at all times.
4. The vehicle' velocity must be monitored at all times.
5. The Brake-by-Wire actuator must be functional and connected to the Safety Controller interface at all times.
6. The Pedestrian Detection sensor must be functional and connected to the Safety Controller interface at all times.

Performance

1. Time loss due to evasive maneuvers should be minimized.
2. Data should be sent from the Pedestrian Detection Sensor to the PCA System every 100 ms.
3. The potential collision time and deceleration needed to avoid said collision must be calculated every 100ms.
4. In the case a pedestrian no longer poses a collision threat, the vehicle is to return to steady state velocity.

2.5 Assumptions and Dependencies

The PCA System's behaviour is dependent on the following hardware, software, environment, and user interaction assumptions.

Hardware

1. Brake-by-wire actuator is working properly.
2. Sensor is a stereo camera that is working properly.
3. Embedded system has no outside access to the system while driving.
4. All hardware components can communicate with one another.
5. The sensor will send data only on pedestrians, not trees or other obstacles.
6. The sensor is either fully functional or off (no faulty data).

Software

1. The PCA System is written as a single algorithm within the vehicle's Safety Controller system.
2. The PCA System is written in C++.
3. The PCA System is written without prior knowledge of the upcoming scenario.
4. The user will use the software as designed and without malicious intent.

Environment

1. The vehicle is presumed to be operating in fair weather.
2. Full visibility of pedestrians is expected.
3. The driving surface is expected to be level.
4. The driving surface is expected to be dry and regular (no ice, gravel, potholes, etc.).

Pedestrian interactions

1. There will only be one pedestrian.
2. The pedestrian can only travel in a straight line at a 90 degree angle to the vehicle.
3. The pedestrian will not go backwards.

User interactions

1. Driver will be alerted when the system is applying brakes.
2. Driver will be alerted when the system is returning to steady state velocity.

2.6 Approportioning of Requirements

The following are requirements that were deemed to be outside the scope of this project, but should be considered for incorporation into the PCA system in the future.

1. The ability to adjust braking in poor weather and irregular road conditions.
2. Ability to turn the vehicle to avoid a collision.
3. Ability to detect and respond to multiple pedestrians.
4. User override capacities.

3 Specific Requirements

The following are specific requirements for the system. The requirements are organized as follows. The first number specifies a more broad requirement, while the second number signifies a more specific requirement within the broader requirement.

1. Monitor the path in front of the vehicle for pedestrians. This includes processing information regarding location, speed, and direction of pedestrians.
    - 1.1 Calculate the distance between the front bumper and anything in front of the car every 100ms.
    - 1.2 Identify potential collisions with pedestrians.
    - 1.3 Notify users of potential collisions.
    - 1.4 Notify pedestrians of potential collisions.
    - 1.5 Identify when a pedestrian no longer constitutes a collision threat.
2. Engage the Brake-by-Wire system in response to a potential collision.
    - 2.1 Notify users when Brake-by-Wire system is engaged.
    - 2.2 Notify users when Brake-by-Wire system is disengaged.
3. Disengage the Brake-by-Wire system when a collision is no longer imminent, allowing the vehicle to return to steady state velocity.
4. The algorithm should avoid collision in all 10 scenarios it is presented with.
5. The algorithm must remain functional (i.e. avoid collisions and minimize lost time) in fail operational mode. This means increasing response time for requested deceleration from 200ms to 900ms.
6. Minimize system vulnerabilities to security breaches.
    - 6.1 Notify the user if the system is turned off for security reasons.
    - 6.2 Shut down system if malfunctions or invalid data are detected.
7. Optimize the avoidance measures to avoid collisions with the least amount of lost time.

4 Modeling Requirements

The following section includes the various models used to explain the PCAS. There is an explanation of what the model is, how it is used, and then shows the specific diagram itself.

4.1 Use Case Diagram

A use case diagram is used to describe the key functions and actors in a system as well as the interactions between those entities. The key functions or 'use cases' are denoted by ovals with a brief text description. Each use case should correspond to at least one of the enumerated requirements from Section 3. Actors are represented by a stick figure in the diagram. Actors are defined as anyone or anything (i.e. sensors or actuators) that interact with a system function. They can both give inputs to the system and expect outputs as well. The communication of an actor with a use case is indicated by a line from the actor to the use case it is participating in. Another important attribute of a use case diagram is the system boundary. The system boundary defines what the system is composed of and thus what its key functions should be. The boundary is represented by the rectangle encasing the use case ovals.

Other important notations in the use case model are the 'include' and 'extends' relationships between use cases. These are denoted using dotted arrows labeled either 'include' or 'extends' respectively. In the case of the 'include' relationship, the use case being pointed to is a part of or is 'used' by the use case at the origin of the arrow. The 'extends' relationship on the other hand implies a special or 'extended' case of a more general use case. In that relationship, the use case being pointed to is the one being extended by the special case at the other end of the arrow.

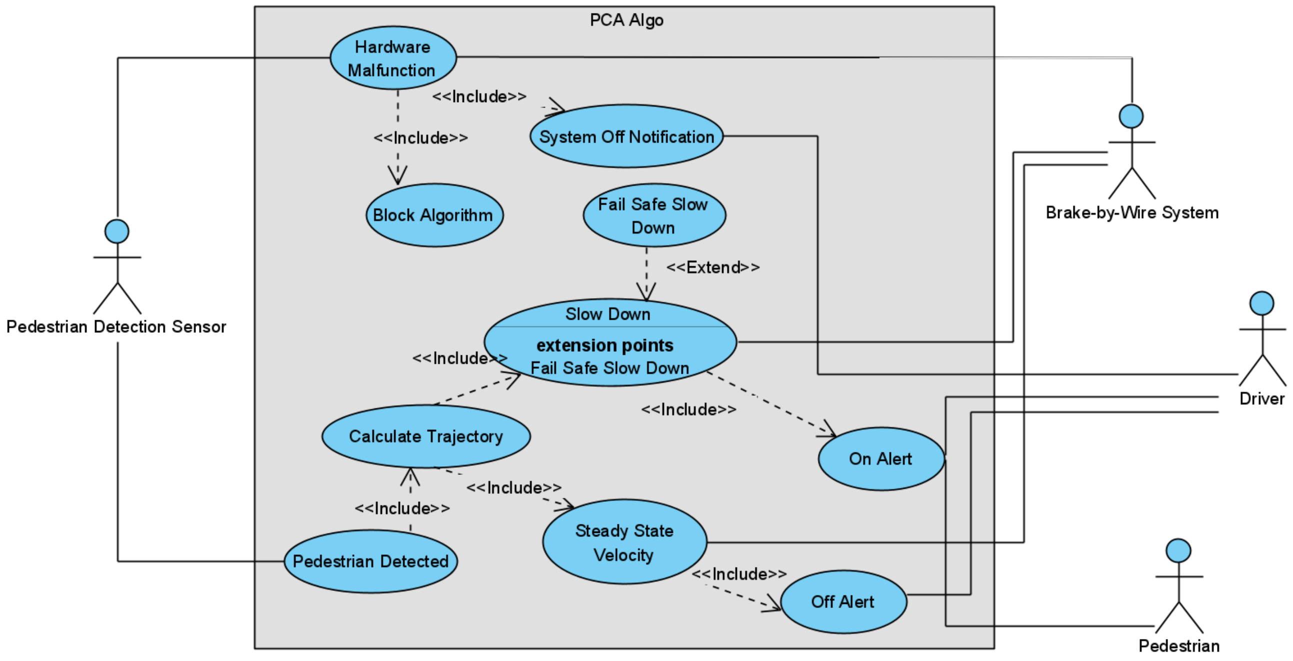


Fig. 2 PCA Use-Case Diagram

These tables explain different scenarios shown in the use case diagram for clarification purposes. Each table includes a use case, actors (components at work in the use case), a description, type of use case (primary or secondary), includes or extends other use cases, cross-references to other use cases, and use cases that may be activated prior.

Use Case:	Pedestrian Detected
Actors:	Pedestrian Detection Sensor (initiator)
Description:	When the vehicle's pedestrian sensor detects a pedestrian, it will provide the PCA algorithm with the pedestrian's (x,y) location relative to the vehicle with an accuracy +/- .5m as well as the pedestrian's velocity +/- .2 m/s, +/- 5 degrees (relative to vehicle). The algorithm will use these inputs to calculate the vehicle and pedestrian's trajectory to determine what avoidance maneuvers, if any, are needed to avoid collision.
Type:	Primary and essential
Includes:	Calculate Trajectory
Extends:	N/A
Cross-refs:	1
Use cases:	N/A

Use Case:	Calculate Trajectory
Actors:	System (initiator)
Description:	When a pedestrian is detected, the system calculates if the vehicle and pedestrian have come within the minimum specified distance of each other. If the vehicle and pedestrian are within that minimum distance of each other, the minimum deceleration needed to avoid collision will be calculated, and the Brake-by-Wire system will be engaged. If the vehicle and pedestrian are not within the minimum distance of each other, the system will ensure the vehicle maintains its steady state velocity, this could mean disengaging the Brake-by-Wire system.
Type:	Primary and essential
Includes:	Slow Down, Steady State
Extends:	N/A

Cross-refs:	1.1, 1.2, 1.5, 2, 3, 4, and 7
Use cases:	Pedestrian Detected

Use Case:	Slow Down
Actors:	System (initiator), Brake-by-Wire System
Description:	When the vehicle has entered within the minimum distance of a detected pedestrian, the Brake-by-Wire system will be engaged to avoid collision.
Type:	Primary and essential
Includes:	On Alert
Extends:	N/A
Cross-refs:	2 and 4
Use cases:	Pedestrian Detected, Calculate Trajectory

Use Case:	Fail-Safe Slow Down
Actors:	System (initiator), Brake-by-Wire System
Description:	When the vehicle is in fail safe mode and has entered within the minimum distance of a detected pedestrian, the Brake-by-Wire system will be engaged with the fail-safe deceleration constraints to avoid collision.
Type:	Extended
Includes:	On Alert
Extends:	Slow Down
Cross-refs:	2, 4 and 5
Use cases:	Pedestrian Detected, Calculate Trajectory

Use Case:	On Alert
Actors:	System (initiator), Pedestrian, Driver
Description:	When the vehicle has entered within the minimum distance of a detected pedestrian, both the driver and pedestrian will be alerted. The driver will receive a notification from inside informing them that the Brake-by-Wire system has been engaged to take avoidance action. The pedestrian will be subtly notified through dimly flashing the vehicle's headlights.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	1.3 and 1.4
Use cases:	Pedestrian Detected, Calculate Trajectory, Slow Down

Use Case:	Steady State Velocity
Actors:	System (initiator), Brake-by-Wire System
Description:	When a pedestrian is detected by the Pedestrian Detection Sensor, but the pedestrian and vehicle are not within the specified minimum distance of each other, the vehicle will maintain or return to steady state velocity.
Type:	Primary and essential
Includes:	Off Alert
Extends:	N/A
Cross-refs:	3 and 7
Use cases:	Pedestrian Detected, Calculate Trajectory

Use Case:	Off Alert
Actors:	System (initiator), Driver

Description:	When the system has calculated that it no longer needs to take avoidance maneuvers to avoid collision with a pedestrian, it will tell the Brake-by-Wire system to return to steady state velocity, notifying the Driver that avoidance maneuvers are no longer necessary and collision has been avoided.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	2.2
Use cases:	Pedestrian Detected, Calculate Trajectory, Steady State Velocity

Use Case:	Hardware Malfunction
Actors:	Pedestrian Detection Sensor (initiator), Brake-by-Wire System (initiator)
Description:	If either the Pedestrian Detection Sensor or the Brake-by-Wire system malfunction or the system senses that the data the Pedestrian Detection Sensor is sending is corrupt, the collision avoidance algorithm will be blocked from running and the Driver will be notified that the collision avoidance system has been turned off.
Type:	Primary and essential
Includes:	Block System
Extends:	N/A
Cross-refs:	6, 6.1, and 6.2
Use cases:	N/A

Use Case:	Block System
Actors:	System (initiator)

Description:	If the system deems that a malfunction in either the Pedestrian Detection Sensor or Brake-by-Wire system has occurred, or it has received corrupt data, the collision avoidance algorithm will be blocked from executing.
Type:	Primary and essential
Includes:	N/A
Extends:	N/A
Cross-refs:	6.2
Use cases:	Hardware Malfunction

Use Case:	System Off Notification
Actors:	System (initiator)
Description:	If the collision avoidance algorithm has been blocked from executing, the user will be notified that the collision avoidance system has been turned off.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	6.1
Use cases:	Hardware Malfunction

4.2 Domain Model

A domain model is a high-level class diagram that describes the key elements in a system and the relationships between those elements. Each rectangle represents a key element or ‘class’, in software engineering terms, of the system. Each class has attributes and operations. Attributes are defining features of a key element. For example, a vehicle could be a class, and some of the attributes of a vehicle would be its velocity, weight, or acceleration. A class’s operations are actions that the key element does. Continuing with the vehicle example, a vehicle can speed up or stop. The attributes of a class are defined in the top part of the rectangle and its operations are defined in the bottom part, below the dividing line.

There are 3 distinct relationships that key elements can have with each other: inheritance (generalization), association, or aggregation. Associations are conceptual links between classes and denoted by a solid line. Associations can be given labels to more clearly define the link between two classes as well as numeric values denoting the number of class objects than can be involved in a single association. Aggregations are a special type of association and denote an ‘is-part-of’ relationship. They can be identified by a diamond symbol on the end of an association link. The last type of relationship, inheritance, is denoted by an arrow between two classes and represents a special case or extension of a particular class.

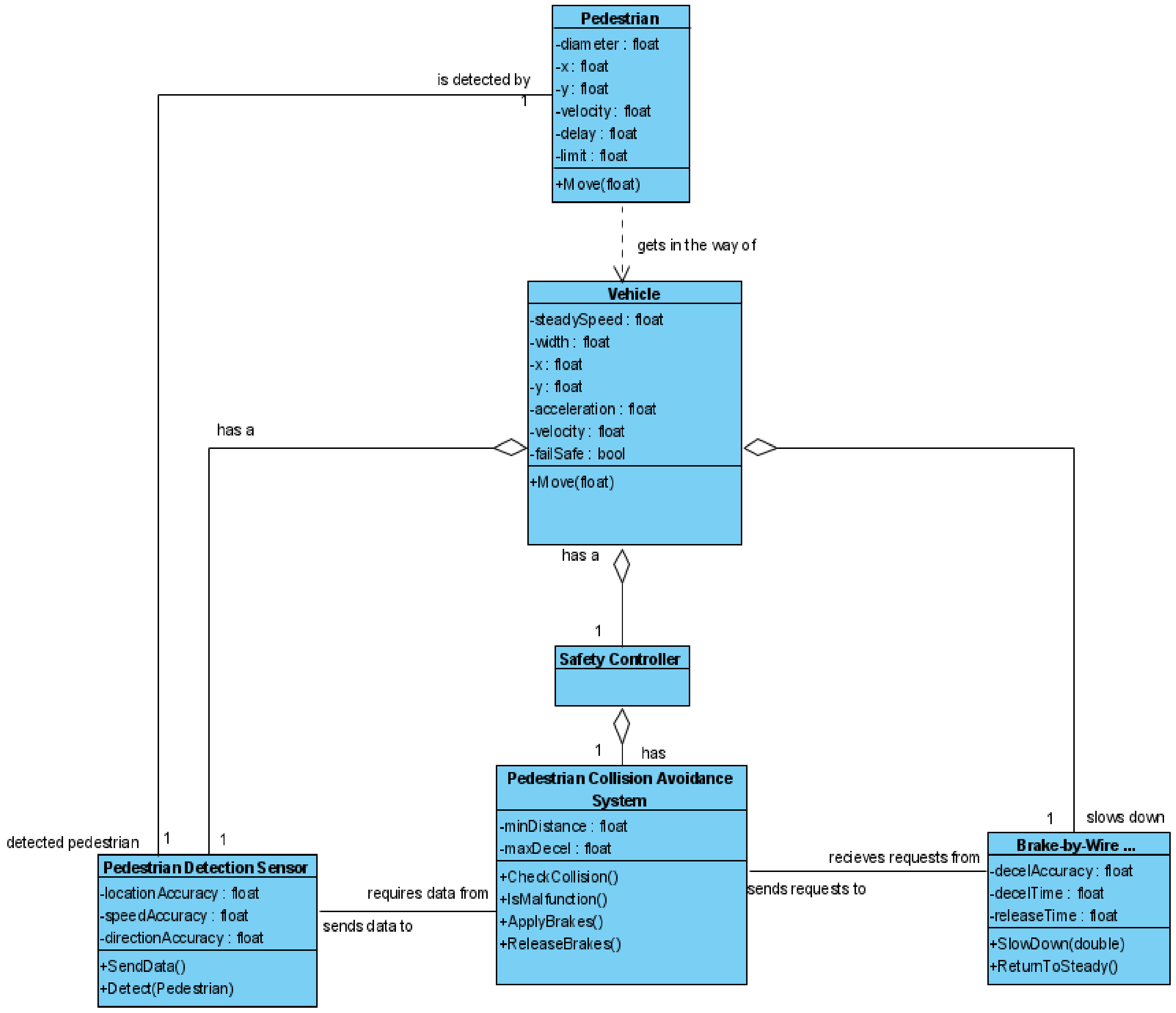


Figure 3. PCA Domain Model

Element Name	**Description**

Brake-by-Wire Actuator		The Brake-by-Wire Actuator responds to deceleration requests by interrupting the steady state velocity control and then applying brake torque via electro-mechanical actuators at all four wheels of the vehicle.
Attributes		
	decelAccuracy: double	For our modeling purposes, the Brake-by-Wire Actuator is able to apply a specified deceleration value within an accuracy of +/- 2%.
	decelTime: double	It takes 200 ms to reach a requested deceleration value.
	releaseTime: double	It takes 100 ms to get back to the acceleration to steady state value.
Operations		
	SlowDown()	Adjusts the vehicle's acceleration to slow it down.
	ReturnToSteady()	Adjusts the vehicle's acceleration to return it to its steady state velocity.
Relationships	A Vehicle has a Brake-by-Wire Actuator. The Brake-by-Wire Actuator receives requests from the Pedestrian Collision Avoidance System.	
UML Extensions	N/A	

Element Name		**Description**
Pedestrian		A person walking along a road.
Attributes		
	X: double	x-coordinate position
	Y: double	y-coordinate position
	Diameter: double	A pedestrian is modeled as being a circle with a diameter of .5 m.

	Velocity: double	The speed and direction at which the pedestrian moves along the y-axis.
	Acceleration: double	The rate at which a pedestrian’s speed is changing.
Operations		
	Move()	This function is used to model a pedestrian moving. It updates the pedestrian’s position based on the amount of time that has passed.
Relationships	A Vehicle has a Brake-by-Wire Actuator. The Brake-by-Wire Actuator receives requests from the Pedestrian Collision Avoidance System.	
UML Extensions	N/A	

Element Name		**Description**
Pedestrian Collision Avoidance System		The Pedestrian Collision Avoidance System is an algorithm that is responsible for enacting avoidance maneuvers in the event of a potential collision with a pedestrian.
Attributes		
	minDistance: double	The closest distance that the algorithm will allow the vehicle to get near a pedestrian.
	maxDecel: double	The maximum deceleration of the vehicle is 0.7 g.
Operations		

	CheckCollision()	This function is the meat and potatoes of the Pedestrian Collision Avoidance System. It will take in the information about a pedestrian sent from the Pedestrian Detection Sensor and determine if avoidance action needs to be taken. If action does need to be taken, the function will issue an ApplyBrakes() request.
	IsMalfunction()	The function that is responsible for disabling the Pedestrian Collision Avoidance System in the event that the Pedestrian Detection Sensor malfunctions, the Brake-by-Wire Actuator malfunctions, or the data sent by the Pedestrian Detection Sensor is determine to be corrupt.
	ApplyBrakes()	This function sends a request to the Brake-by-Wire Actuator to apply a given deceleration value.
	ReleaseBrakes()	This function sends a request to the Brake-by-Wire Actuator to return to steady state velocity.
Relationships	The Pedestrian Collision Avoidance System is part of the Safety Controller of a Vehicle.	
UML Extensions	N/A	

Element Name		**Description**
Pedestrian Detection Sensor		The Pedestrian Detection Sensor is a stereo camera that has pedestrian recognition and tracking capabilities.
Attributes		
	locationAccuracy: double	The sensor can give a pedestrian's (x,y) location relative to the car with an accuracy of +/- .5 m.
	speedAccuracy: double	The sensor can give a pedestrian's speed with an accuracy of +/- .2m/s.

	directionAccuracy: double	The sensor can give a pedestrian’s direction with an accuracy of +/- 5 degrees.
Operations		
	SendData()	This function sends a detected pedestrian’s relative location to the vehicle as well as their velocity to the Pedestrian Collision Avoidance System.
Relationships	The Pedestrian Detection Sensor detects a pedestrian and sends the data about that pedestrian to the Pedestrian Collision Avoidance System.	
UML Extensions	N/A	

Element Name		**Description**
Safety Controller		The Safety Controller is a large system within the vehicle that is responsible for the vehicle’s safety operations.
Attributes		
Operations		
Relationships	The Safety Controller is part of the Vehicle.	
UML Extensions	N/A	

Element Name		**Description**
Vehicle		A road vehicle with 4 wheels powered by an internal combustion engine or electric motor and able to carry a small number of people.
Attributes		
	steadySpeed: double	For our modeling purposes, a vehicle has a normal steady state speed of 50 kph (13.9 m/s).

	width: double	For our modeling purposes, a vehicle has a width of 2 m, which marks its collision zone.
	x: double	x-coordinate position
	y: double	y-coordinate position
	Acceleration: double	The rate at which a vehicle's speed is changing. For our modeling purposes a vehicle can have a maximum acceleration of .25 g and a maximum deceleration of 0.7g (1 g = 9.81 m/s^2).
	Velocity: double	The speed and direction at which the vehicle moves along the x-axis.
	failSafe: bool	A vehicle has a fail safe mode in which the response time to reach a requested deceleration value is increased. This flag indicates whether a vehicle is in fail safe mode.
Operations		
	Move()	This function is used to model a vehicle moving. It updates the vehicle's position based on the amount of time that has passed.
Relationships	A vehicle has a Pedestrian Detection Sensor, Safety Controller, and Brake-by-Wire Actuator.	
UML Extensions	N/A	

4.3 Sequence Diagrams

Sequence diagrams are an interaction diagram that maps out how processes occur and the order in which functions are called. Each rectangle lining the top of the diagram represents an instance, or object, of a relevant class. The vertical line going downwards from the rectangle is the object's lifeline. The thin rectangles on the lifeline represent when the object is active. For instance, the vehicle is on for the entirety of the scenario whereas the brake actuator is only active whenever the vehicle needs to accelerate or decelerate.

Arrows represent functions or operations occurring during the scenario. Arrows between objects mean a function involving another object is being used. Arrows originating at and pointing to the same object mean an internal function or operation is running. The dotted arrow means the function completes and exits the current object.

4.3.1 Stationary Pedestrian Scenario

In this scenario, the pedestrian is stationary in the path of the vehicle. The vehicle detects the pedestrian and slows to a stop. Since the pedestrian never moves out of the path during the scope of the scenario, the vehicle is also stopped for the rest of the duration.

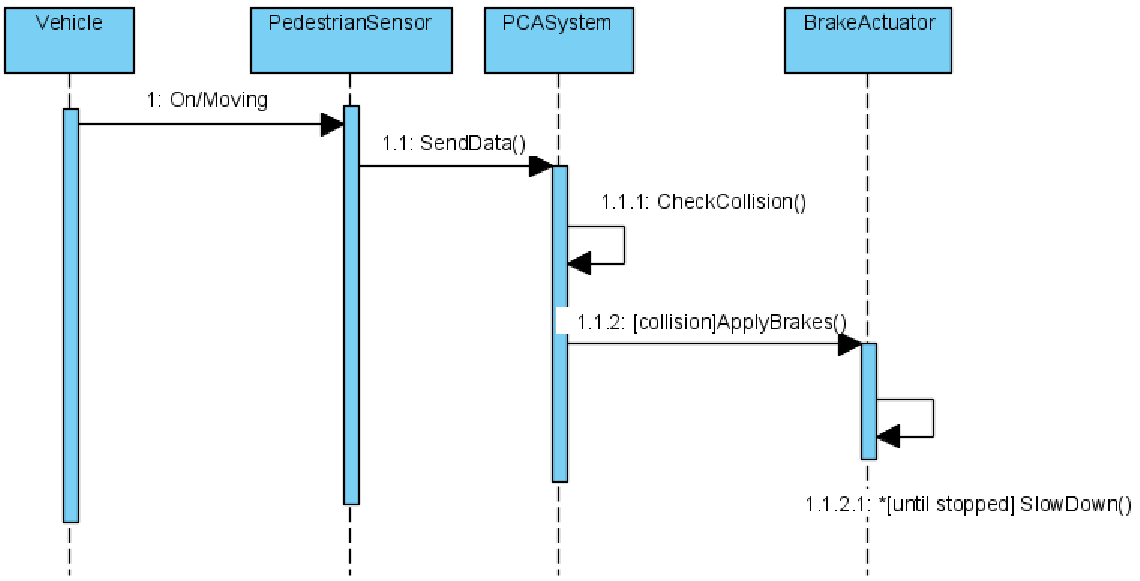


Fig. 4 PCA Sequence Diagram: Stationary Pedestrian

4.3.2 Out of Path Pedestrian Scenario

In this scenario, there is no collision chance between the vehicle and the pedestrian. This can be because the pedestrian is standing out of the way of the vehicle’s path or because it crosses the road before the vehicle gets close enough to detect it. In this case, the vehicle runs the entire duration at steady state velocity, and the brakes never kick in.

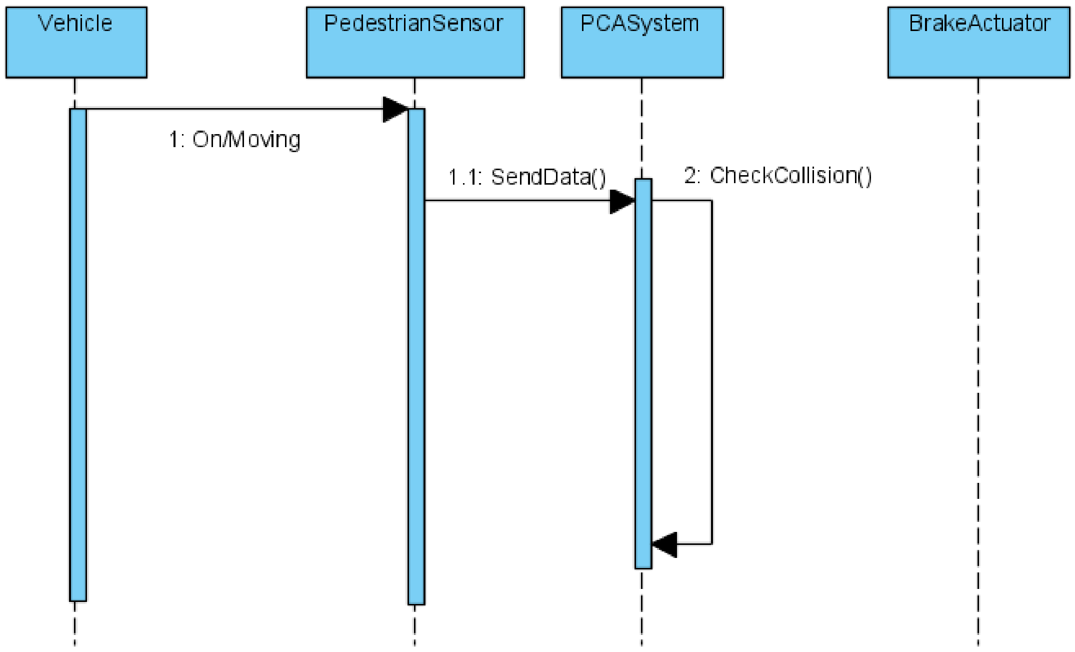


Fig. 5 PCA Sequence Diagram: No Collision Threat

4.3.3 Moving Pedestrian Scenario

In this scenario, the pedestrian is moving when the vehicle's sensor detects it. The vehicle slows or stops in front of the pedestrian and then returns to steady state velocity when the pedestrian is no longer in range. This means that initially, the brakes are applied, but they are later lifted.

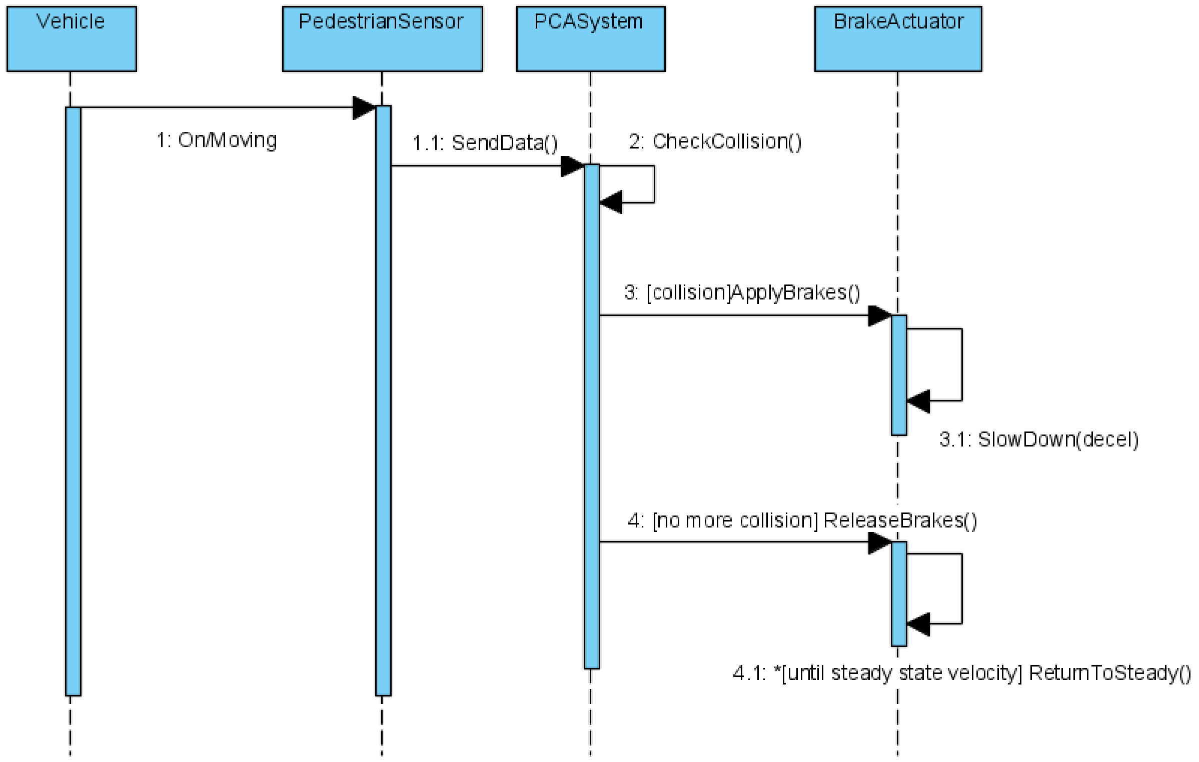

Vehicle
PedestrianSensor
PCASystem
BrakeActuator
1: On/Moving
1.1: SendData()
2: CheckCollision()
3: [collision]ApplyBrakes()
3.1: SlowDown(decel)
4: [no more collision] ReleaseBrakes()
4.1: *[until steady state velocity] ReturnToSteady()

Figure 6. PCA Sequence Diagram: Pedestrian Moved out of the Way

4.4 State Diagram

A state diagram is used to visualize a system's behavior through each element involved. Each of the state diagrams below corresponds to a key element of the system. The key elements for this system are the vehicle, the PCA system, the Brake-by-Wire actuator, and the pedestrian detection camera sensor. The black dot represents the initial pseudo state in every state diagram. Each of the rectangles in the state diagrams below are called states and represent the condition or status of a key element at a given moment. An example of a state is the "monitoring" state in the diagram for the PCA system. "Monitoring" means that the PCA system is actively receiving data from the camera sensor to check if any pedestrians are in the path of the vehicle. The arrows from one element to another are the transition events that cause a change from one state to another. An example of a transition is a "hardware fault". When a hardware fault occurs in the PCA system, it causes an alert to the driver which transitions the system to the "Alert" state.

4.4.1 PCA System State Diagram

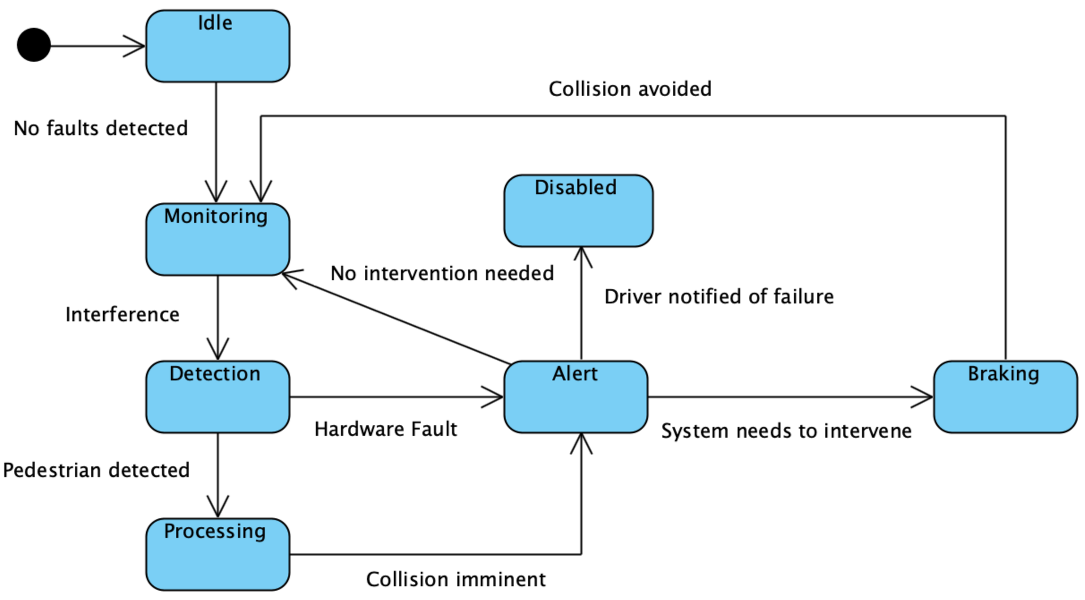


Fig.7 PCA System State Diagram

States	**Description**
Idle	Initial idle state.
Monitoring	System continually receives data from the camera sensor to check if there are any pedestrians.
Detection	The system determines what has caused attention.
Processing	The system makes calculations regarding the pedestrian detected and decides if a collision is imminent.
Alert	The system alerts the driver of either a fault or a possible collision.
Disabled	The system is deactivated due to a malfunction.
Braking	The system intervenes and applies the brakes to avoid a collision with the pedestrian.
Transitions	**Description**
No faults detected	System is operational without any issues.
Interference	An interruption occurs and must be checked.

Pedestrian detected	Pedestrian data from the camera sensor has been received.
Collision Imminent	Collision is possible based on calculations.
Hardware Fault	System fault detected.
System needs to intervene	System needs to apply brakes to prevent collision.
No intervention needed	Risk of collision no longer exists.
Driver notified of failure	Alert given to the driver that the system has been deactivated due to a fault.
Collision avoided	Collision has been avoided successfully.

4.4.2 Vehicle State Diagram

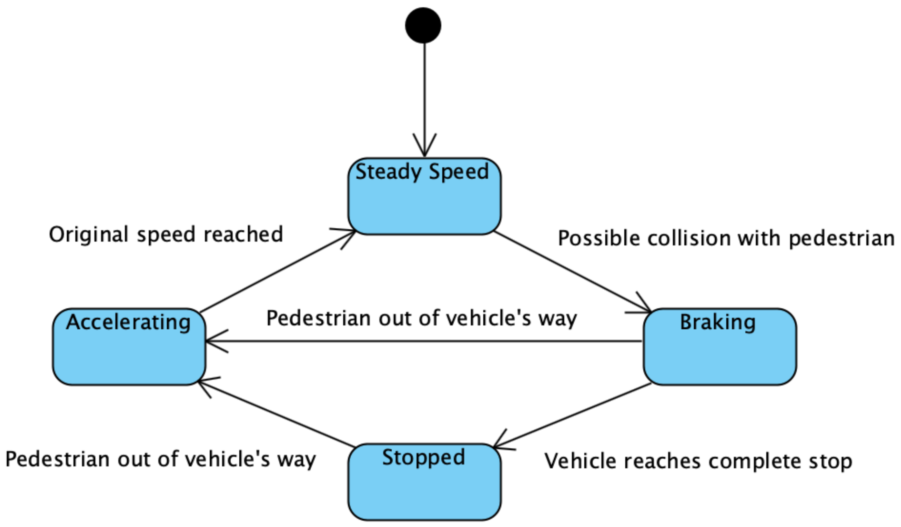

Steady Speed
Original speed reached
Possible collision with pedestrian
Accelerating
Pedestrian out of vehicle's way
Braking
Pedestrian out of vehicle's way
Stopped
Vehicle reaches complete stop

Fig.8 Vehicle State Diagram

States	**Description**
Steady Speed	The vehicle is traveling at a steady rate of speed.
Braking	The vehicle is slowing down after applying the brakes.
Stopped	The vehicle has reached a complete stop after braking.

Accelerating	The vehicle is gaining speed to reach the original steady speed.
Transitions	**Description**
Possible collision with pedestrian	Vehicle has a pedestrian in its path which could lead to a possible collision.
Pedestrian out of vehicle's way	Pedestrian is no longer in the way of the vehicle.
Vehicle reaches complete stop	Vehicle stops completely to avoid collision.
Original speed reached	Vehicle is back up to steady speed.

4.4.3 Pedestrian Detection Sensor State Diagram

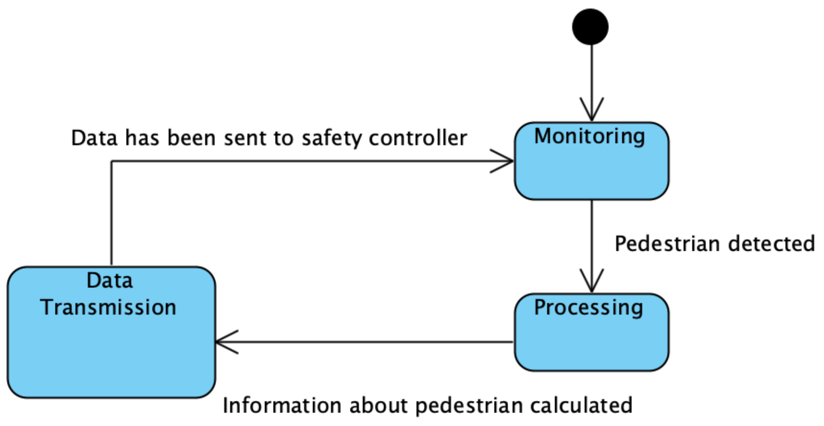

Data has been sent to safety controller
Monitoring
Pedestrian detected
Data Transmission
Processing
Information about pedestrian calculated

Fig.9 Pedestrian Detection Sensor State Diagram

States	**Description**
Monitoring	Camera sensor looks in front of the vehicle to spot any pedestrians in the path of the vehicle.
Processing	Sensor has detected a pedestrian and obtains data about the speed and position of the pedestrian.
Data Transmission	The sensor sends the pedestrian data to the safety controller and algorithm in the PCA system.
Transitions	**Description**
Pedestrian detected	Pedestrian Detection Sensor has detected a pedestrian towards the front of the vehicle.

Information about pedestrian calculated	Pedestrian Detection Sensor has determined the location, speed, and direction of the pedestrian.
Data has been sent to safety controller	The data regarding the pedestrian has been sent to the safety controller for use by the PCA system.

4.4.4 Brake-by-Wire Actuator State Diagram

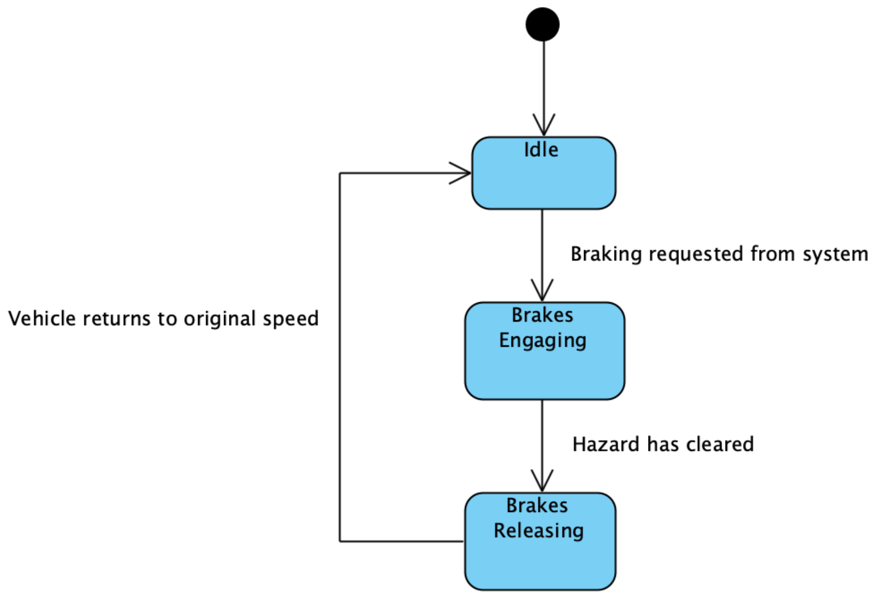

Idle
Braking requested from system
Brakes Engaging
Vehicle returns to original speed
Hazard has cleared
Brakes Releasing

Fig.10 Brake-by-Wire State Diagram

States	**Description**
Idle	The Brake-by-Wire Actuator is inactive until a request has been made.
Brakes Engaging	The Brake-by-Wire Actuator engages the brakes once the PCA system has made a request to do so.
Brakes Releasing	Once the conditions are safe, the brakes are released after a request from the PCA system.
Transitions	**Description**
Braking requested from system	Brake actuator receives request to apply brakes.
Hazard has cleared	Pedestrian is out of the way of the vehicle.
Vehicle returns to original speed	Vehicle goes back to steady speed.

4 Prototype

The prototype is an executable in which the user can observe the behavior of a vehicle running the PCA System under a variety of scenarios. Pedestrian behavior (standing still, moving, or moving after a delay) as well as starting and ending positions vary between scenarios. When the PCA System has detected the possibility of a future collision with a pedestrian, the prototype demonstrates the vehicle's deceleration reaction. In the event that a pedestrian moves and no longer poses a collision threat, the prototype demonstrates how the vehicle will accelerate to steady state velocity. Additionally, the user can observe that if a pedestrian is not within the vehicle's collision zone, there will be no change in vehicle behavior.

4.1 How to Run Prototype

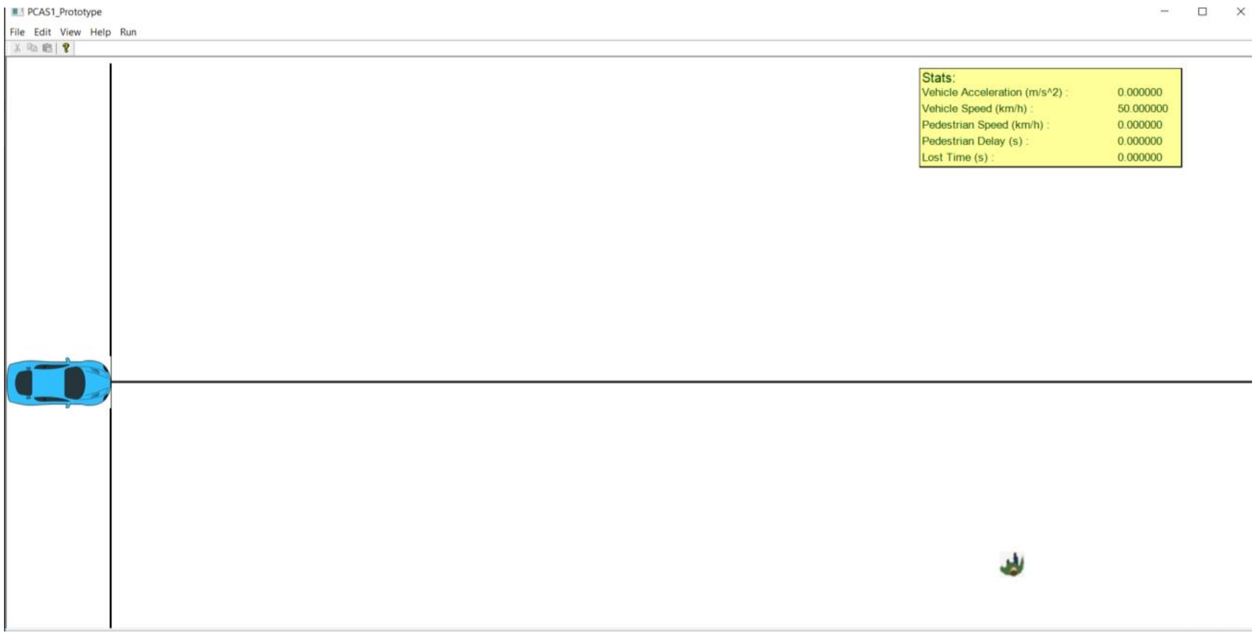
The downloadable executable of the prototype is available as a zipped folder here: https://cse.msu.edu/~chenaiho/prototype.html. Once downloaded and unzipped following the instructions on the webpage, the user can run the executable and select from the predefined scenarios under the "run" menu bar drop down.

The prototype must be run on a Microsoft Windows operating system, version 7 or higher. If access to a device running Windows is unattainable, a video demonstration of the prototype is available on the same webpage for viewing.

The second version of the prototype is intended to be a functional web interface.

4.2 Sample Scenarios

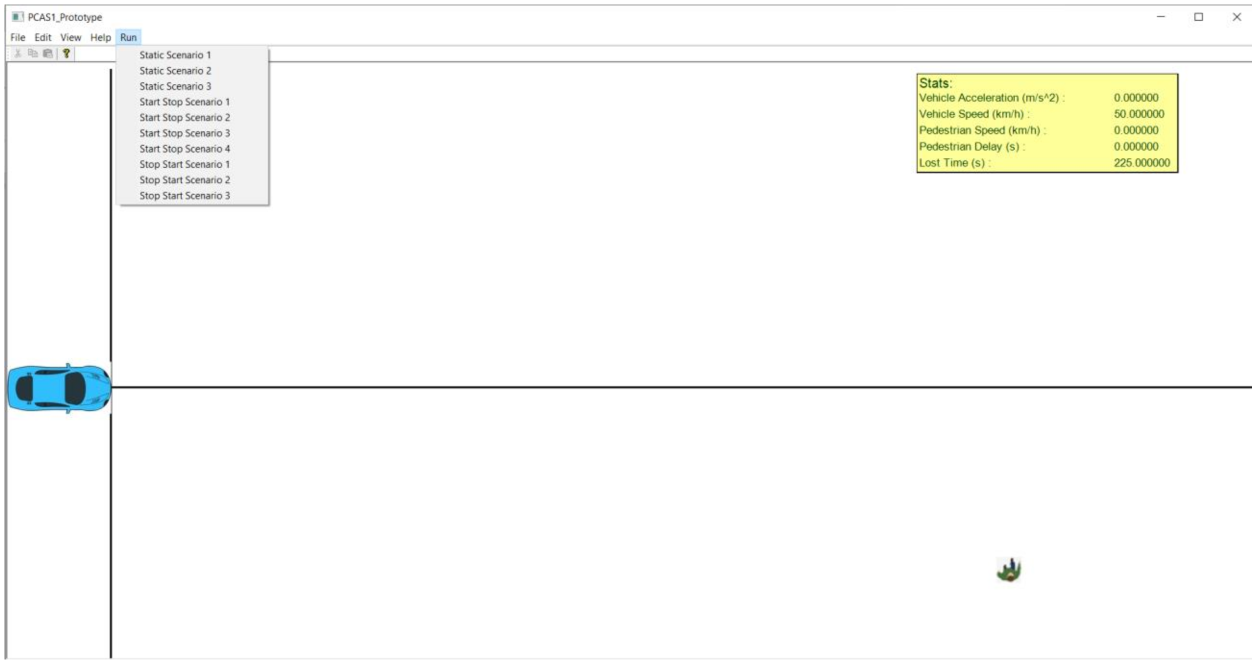
The below image depicts what is shown when the prototype first executes. This is the prototype's default screen. The 2 black lines represent the x and y axis' for each of the scenarios. The x-axis goes from 0 to 50 m while the y axis ranges from -12.5 to 12.5 m. The vehicle image has been scaled to represent a vehicle that is 2 m wide and 4 m long. The pedestrian image has been scaled to model a pedestrian with a diameter of .5 m. The yellow box in the upper right-hand corner serves as a 'stats bar', displaying vehicle acceleration, vehicle speed, pedestrian speed, pedestrian delay, and lost time.



```
Stats:
Vehicle Acceleration (m/s^2) :    0.000000
Vehicle Speed (km/h) :            50.000000
Pedestrian Speed (km/h) :         0.000000
Pedestrian Delay (s) :            0.000000
Lost Time (s) :                   0.000000
```

Fig. 11 Initial Prototype Screen

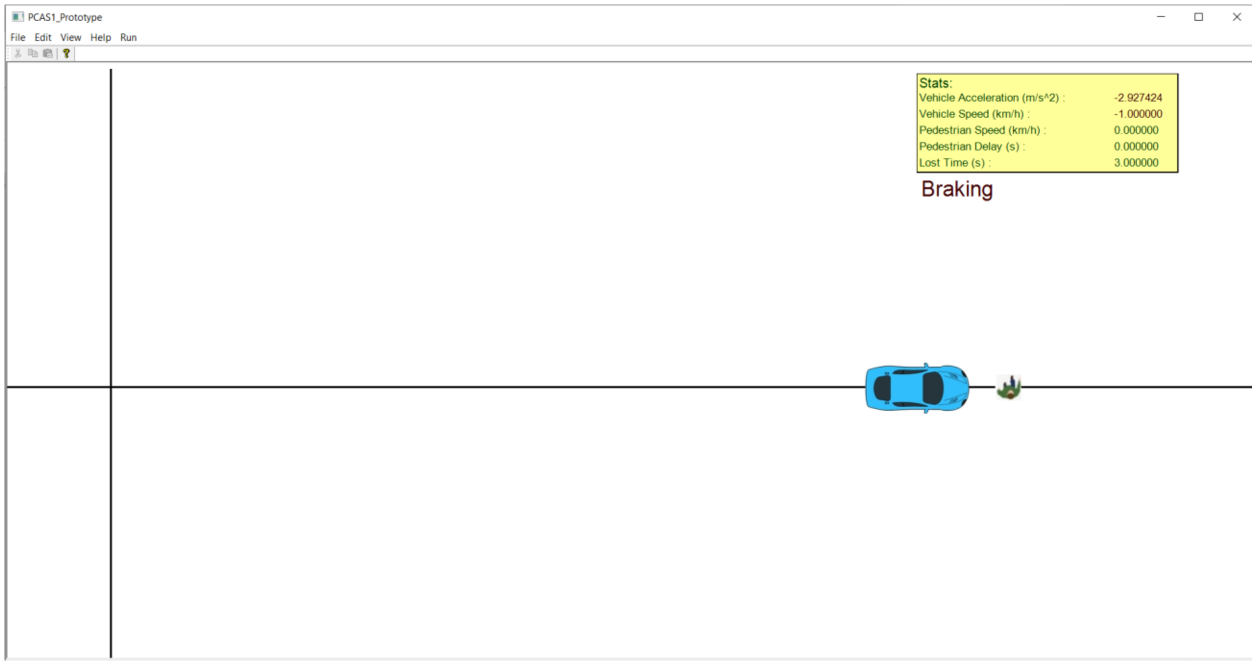
To run a scenario, the user will click the Run menu option. When they do this, a drop-down menu of possible scenarios to run appears.



```
Stats:
Vehicle Acceleration (m/s^2) :    0.000000
Vehicle Speed (km/h) :            50.000000
Pedestrian Speed (km/h) :         0.000000
Pedestrian Delay (s) :            0.000000
Lost Time (s) :                   225.000000
```

Fig. 12 Prototype Scenario Menu

The below image shows Static Scenario 1 executing. For this scenario, the pedestrian's starting position is (35, 0) and they will remain stationary at this position. The vehicle begins at position (0,0) with a velocity of 13.9 m/s. As the vehicle nears the pedestrian, it will begin to decelerate until it eventually comes to a stop in front of the pedestrian.

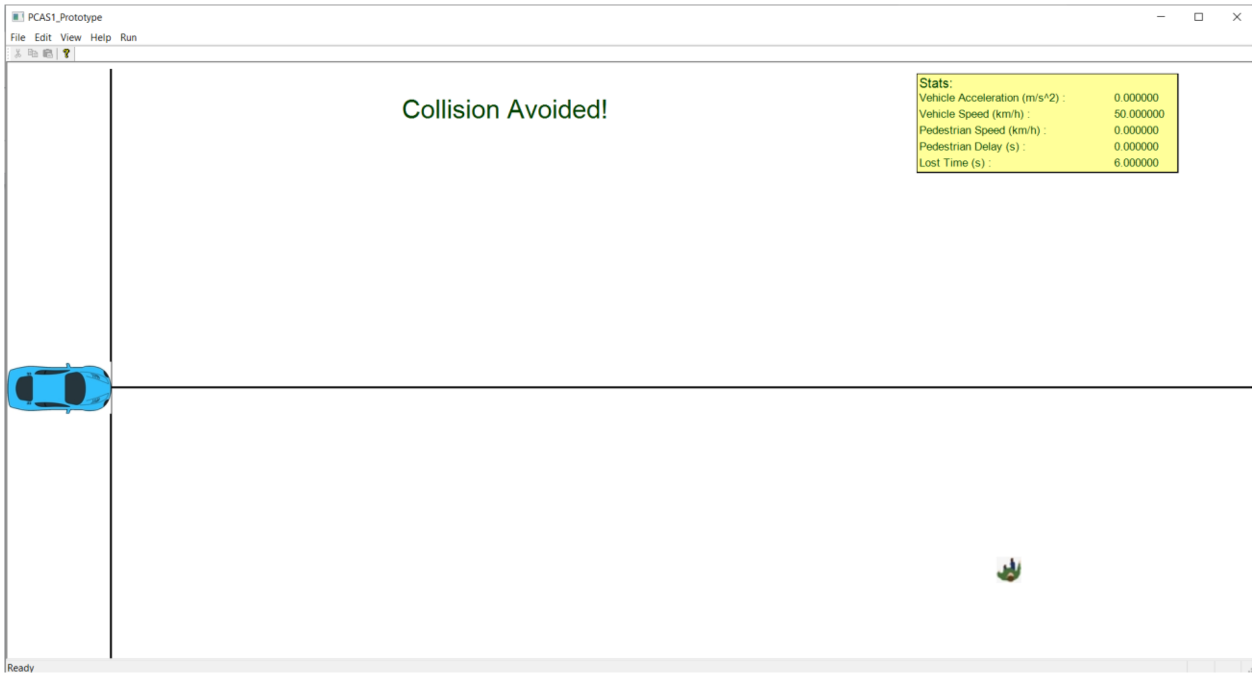


Stats:
Vehicle Acceleration (m/s^2) : -2.927424
Vehicle Speed (km/h) : -1.000000
Pedestrian Speed (km/h) : 0.000000
Pedestrian Delay (s) : 0.000000
Lost Time (s) : 3.000000

Braking

Fig. 13 Executing Static Scenario

When the simulation of the chosen scenario is complete, both the pedestrian and vehicle will reset to their default positions, and the text “Collision Avoided” will appear on the screen. Once this occurs, the user is free to select another scenario to run.



Collision Avoided!

Stats:
Vehicle Acceleration (m/s^2) : 0.000000
Vehicle Speed (km/h) : 50.000000
Pedestrian Speed (km/h) : 0.000000
Pedestrian Delay (s) : 0.000000
Lost Time (s) : 6.000000

Fig. 14 Finished Scenario

5 References

[1] “Automated Vehicle System for Pedestrian Collision Avoidance Function,” October 2020. [Online]. Available: https://cse.msu.edu/~chenaiho/

[2] C. Capaldi, “Automated Vehicle System for Pedestrian Collision Avoidance Function”, Dataspeed, Inc., 2020.

6 Point of Contact

For further information regarding this document and project, please contact **Prof. Betty H.C. Cheng** at Michigan State University (chengb at 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.