

Software Requirements Specification (SRS)

PEDAC1

Authors: Samuel Chung, Christopher Cummings, Wan Kim, Tyler Maklebust, Mark Velez

Customer: Mr. David Agnew, Director Advanced Engineering Mobis NA

Instructor: Dr. Betty H.C. Cheng

1 Introduction

This SRS document provides an overall project description, specific requirements, along with models and a prototype of the proposed solution. The project described is the PEDAC, an automated pedestrian collision avoidance system implemented in fully autonomous vehicles. These vehicles have no drivers--that is, all humans in the vehicle are considered passengers with no interface to interact with the vehicle or its subsystems, including the PEDAC. The PEDAC system will avoid all collisions with pedestrians. It does this by calculating the appropriate speed to maintain in order to avoid possible collisions based on input from a stereo camera system. The system will also maximize efficiency of the vehicle by maintaining the highest velocity possible while still avoiding any potential collisions with pedestrians. All vehicle functions such as acceleration, deceleration, and enabling or disabling cruise control are managed completely by the software in the vehicle. The models section contains diagrams which describe the implementation of the system along with use case diagrams enumerating the use cases that satisfy the goals and requirements of the project.

1.1 Purpose

This SRS document is intended for Mr. David Agnew, Director Advanced Engineering of Mobis NA and its purpose is to provide a detailed description of the requirements of the project and also a detailed technical description of the proposed solution. The PEDAC's objective is to reduce pedestrian collisions to zero in fully autonomous driverless vehicles and maximize the efficiency of the vehicles by minimizing the time it takes to get from one point to another with a set of given scenarios.

1.2 Scope

The software solution proposed is a submodule in fully autonomous driving systems that is responsible for responding to pedestrian hazards. Pedestrian hazards are defined as any pedestrians that could potentially place themselves within the trajectory of the vehicle and pose a threat to the safety of both themselves and the passengers in the autonomous vehicle. PEDAC seeks to maximize safety and efficiency in automated driving systems in which it is included, allowing autonomous vehicles to be both safer and more efficient than human-operated vehicles. It will be implemented as embedded software in fully automated driving systems. The PEDAC system will determine whether

the Brake-By-Wire System should be activated or if the steady-state velocity can be maintained by analyzing the collision path between the vehicle and any detected pedestrians.

1.3 Definitions, acronyms, and abbreviations

Definitions:

PEDAC: Automated Pedestrian Collision Avoidance System

Pedestrian Hazard: a pedestrian determined to be currently in the collision path or projected to be

Steady State Velocity: a constant velocity to be maintained when there are no Pedestrian Hazards

Cruise Control: the vehicle state in which the vehicle maintains a steady state velocity

Cruise Control System: system which maintains the vehicle's Steady State Velocity

Brake-By-Wire System: system which manages the brakes in the autonomous driving system

Vehicle Throttle: the servomechanism which regulates the vehicle speed

Fail Safe Mode: an operational mode that accounts for situations where additional time is required for vehicle deceleration (900 ms to reach requested deceleration, instead of 200 ms)

Stereo Camera Array: an array of sensors at used to determine the location of any pedestrians

CAN Bus: a vehicle bus standard that allows devices to communicate without a host computer

Safety Controller: system which manages and syncs the Vehicle Throttle and Brake-By-Wire System

Units:

kph: kilometers per hour

m: meters

m/s: meters per second

m/s²: meters per second squared

g (acceleration): 9.81m/s²

Abbreviations:

CAN: Control Area Network

1.4 Organization

The rest of this document contains detailed descriptions of the product including its functions, behaviors, dependencies, and requirements in figures, diagrams, and prose.

The structure of the following sections, along with their section headers, is as follows:

- 2.0 Overall Description
 - Descriptions of the product's context, functions, constraints, and requirements
- 2.1 Product Perspective
 - A description of the product's context and how it fits into the larger system
- 2.2 Product Functions
 - A description of the product's functions and high-level goals
- 2.3 User Characteristics
 - A description of the user of the system and the user's expectations
- 2.4 Constraints
 - A low level description of all system constraints and critical properties
- 2.5 Assumptions and Dependencies
 - Descriptions of all assumptions made in the design of the system and its dependencies
- 2.6 Apportioning of Requirements
 - Descriptions of requirements which fall outside the scope of the project
- 3.0 Specific Requirements
 - Low level detailed descriptions of all project requirements
- 4.0 Modeling Requirements
 - Use case, class, sequence, and state diagrams of the proposed system
- 5.0 Prototype
 - Description of the solution prototype
- 5.1 How to Run Prototype
 - Instructions on executing the prototype
- 5.2 Sample Scenarios
 - Enumerates all simulated scenarios for the prototype and their results
- 6.0 References
 - Lists all referenced documents and resources with links
- 7.0 Point of Contact
 - Contact information for further information about the document

2 Overall Description

This section covers the overall descriptions of the project and the proposed solution. This includes the product perspective, product functions, user characteristics, constraints, assumptions and dependencies, and the agreed requirements. An overview of each section and the content it contains are as follows:

Product Perspective:

Describes the context of the product and how it fits into larger systems

Product Functions:

Summarizes all major functions to be performed by the system

User Characteristics:

Describes any users of the system and how they interact with it

Constraints section:

Describes all system constraints and safety-critical properties

Assumptions and Dependencies:

Describes all assumptions made in the design of the system and lists dependencies

Apportioning of Requirements:

Describes all requirements determined to be outside of the scope of the project

2.1 Product Perspective

PEDAC is an embedded system in autonomous vehicles that recognizes and avoids collision with pedestrian hazards by analyzing the collision path between the vehicle and any detected pedestrians. Both the PEDAC system and the vehicle system are fully autonomous: there is nobody in the driver's seat of the vehicle, and there are no options to override any of the PEDAC functions. After calculating the collision paths and determining whether the car can safely maintain Steady State Velocity or requires braking to maintain a safe distance to the pedestrian, PEDAC sends deceleration messages to the Safety Controller, which manages the Brake-By-Wire System and the Vehicle Throttle. Conversely, if the system is below the Steady State Velocity and PEDAC determines that it is safe to accelerate back up to Steady State Velocity, it will send acceleration signals to the Safety Controller. The acceleration and deceleration of the vehicle are not managed by the PEDAC System. The Steady State Speed of the vehicle is managed by the Vehicle Throttle, which can be overridden by the Brake-By-Wire System. This means that PEDAC has no direct interface with the vehicle acceleration or deceleration, but relies on the Safety Controller to interpret the messages and send the appropriate signals to the Brake-By-Wire System or the Vehicle Throttle to accelerate, decelerate, or maintain the Steady State Velocity.

System Interfaces:

PEDAC <-> Safety Controller
Safety Controller <-> Brake-By-Wire System
Safety Controller <-> Vehicle Throttle

The PEDAC system communicates with the Safety Controller, sending instructions to accelerate or decelerate after analyzing the vehicle collision path and any Pedestrian Hazards. The Safety Controller, which has full override capability to the state of the vehicle, interprets these instructions and sends the appropriate signals to the Brake-By-Wire System or the Vehicle Throttle.

Hardware Interfaces:

PEDAC <-> Stereo Camera Array
Brake-By-Wire System <-> Brakes
Safety Controller <-> Vehicle Throttle

The PEDAC system reads directly from the Stereo Camera Array and calculates the collision path between the vehicle and any detected pedestrians. The Brake-By-Wire System interfaces directly with the vehicle brakes to decelerate the vehicle. The Cruise Control System interfaces directly with the Vehicle Throttle, maintaining the Steady State Speed.

Software Interfaces:

PEDAC <-> Safety Controller
Safety Controller <-> Brake-By-Wire System
Safety Controller <-> Vehicle Throttle

The PEDAC system can invoke functions on the Safety Controller, which in turn invokes functions on the Brake-By-Wire System or the Cruise Control System based on the function invoked. As mentioned above, the PEDAC system does not directly interface with the Brake-By-Wire or Cruise Control submodules. It communicates with the Safety Controller, which communicates with the Brake-By-Wire System and the Cruise Control System.

Communication Interfaces:

PEDAC <-> CAN Bus
Safety Controller <-> CAN Bus
Brake-By-Wire System <-> CAN Bus
Stereo Camera Array <-> CAN Bus
Brakes <-> CAN Bus
Vehicle Throttle <-> CAN Bus

All systems and devices in the vehicle communicate with each other through the CAN Bus, which is a vehicle bus standard that allows all devices in a vehicle to communicate with each other without a host computer. All messages, sensor data, and control data are sent through the CAN Bus.

2.2 Product Functions

The PEDAC system will comprise of the following main functions:

1. Monitor path in front of vehicle while driving, looking for pedestrians and identifying potential collisions with them.
2. Determine potential collisions by analyzing collision path between vehicle and pedestrian.
3. Take action to avoid pedestrians by executing velocity reduction commands (automatic braking) which override the current Steady State Velocity of the vehicle.
4. Vehicle velocity will automatically return to Steady State Velocity when the hazard no longer exists
5. Optimize efficiency of travel time by minimizing the time lost when avoiding pedestrian collisions
6. Fail Safe mode which can be activated before a given scenario, where the algorithm will account for an increase in time to reach requested deceleration
7. If collision cannot be avoided, apply the maximum amount of deceleration to mitigate as much of the impact as possible

2.3 User Characteristics

The users of the PEDAC system are simply passengers in the vehicle. The PEDAC system does not contain an interface through which a human user can interact with its function. The vehicles described in the project have no drivers, and all humans in the vehicle are defined to be passengers. The vehicle fully manages its own state and there is no option for the passengers override any of the vehicle functions.

2.4 Constraints

The primary constraints of the PEDAC system are the vehicle acceleration and deceleration limits. Acceleration of the vehicle is limited to 0.25g (2.45m/s^2) and maximum deceleration from braking is 0.7g (6.87m/s^2). There is also a delay of 200ms to reach the requested deceleration amount and a delay of 100ms to completely release brakes. Along with change in speed, maximum speed (steady state speed) is limited to 50kph (13.9m/s).

The system is also constrained by the output rate of the camera sensors. Information can only be retrieved from the sensor output packets every 100 milliseconds, which limits how quickly the PEDAC system can react to changing conditions.

2.5 Assumptions and Dependencies

This section describes the assumptions made in the design of the PEDAC system and its dependencies. One assumption made about the vehicle system is that the maximum possible deceleration for the vehicle is .7g under all conditions. If the vehicle cannot slow down at this rate, the core requirement of the system, not hitting a pedestrian, might not hold true. Another assumption is that the pedestrians can never move at a speed greater than the speed defined, 10kph. The system algorithm will be based on the assumption that pedestrians cannot exceed this speed.

In order for the PEDAC system to function properly, other systems which it depends on must be functional as well. The camera system, along with the ability to accurately identify pedestrians and their locations, are dependencies of the algorithm. In the design of this system, an assumption that the camera is always accurate within +/- 0.5m was made. Pedestrians must be accurately identified in order for the system to respond accordingly. The Brake-By-Wire System and the Vehicle Throttles are also vital dependencies in the PEDAC system; these are the only available means of avoiding a collision and minimizing time lost per the project requirements and must be functional in order for the algorithm to meet the requirements.

2.6 Apportioning of Requirements

There are some requirements that fall outside the scope of this project. The implementation of pedestrian identification is to be handled by the camera sensors, but this would likely be part of the PEDAC system in a production solution. The effect of lighting or any other conditions on pedestrian detection will not be taken into account for this project, either. Factoring in the effect of weather on braking efficiency is also outside the scope of this project, deceleration will be assumed to be effective regardless of any conditions.

3 Specific Requirements

The specific requirements of the PEDAC system are defined here.

The scenarios defined in the project are described below:

The vehicle always starts at the point $(x,y) = (0,0)$ and moves at 50 kph (13.9 m/s) in the +x direction. Pedestrian always starts at the point $(x,y) = (35,-7)$. The following scenarios describe pedestrian movements.

Moving then stopped				
Scen #	Initial Position, Y_i	End Position, Y_f	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				
Scen #	Initial Position, Y_i	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				
	(m)		(kph)	(kph)
8	0	NA	0	0
9	-2	NA	0	0
10	-4	NA	0	0

Figure 3.1

Benchmarks for the quality of the algorithm:

Effectiveness: Avoid all collisions with pedestrians, given the above scenarios

The functional requirements for effectiveness and its related constraints are outlined below:

1. Calculating Pedestrian Locations
 - a. Use the stereo camera sensor output, which includes all identified pedestrians in front of the vehicle (180 degrees) and their current position, speed, and direction
 - b. Stereo camera output is sent as a packet every 100ms, this system will handle reception of all packets at this interval.
 - c. To account for delay in camera sensor output and the actual position of the pedestrian, the pedestrian will be assumed to be 0.167m closer to the path of the vehicle. This value is determined based on the 100ms interval and the max pedestrian speed of 1.67m/s.
 - d. The accuracy of the stereo camera sensor is +/- 0.5m. This will be accounted for by assuming the actual pedestrian location is 0.5m closer to the path of the vehicle to ensure all pedestrian collisions are still avoided in the case of maximum inaccuracy of sensor output.
2. Avoid all collisions regardless of pedestrian movement
 - a. Worst-case scenario would have the pedestrian moving directly towards the vehicle path at maximum speed, which has been defined as 6kph (1.67m/s), or not moving if they are already in the path of the vehicle
3. Check identified pedestrians from above for any possible collisions (Pedestrian Hazards)
 - a. Find the possible collision point by checking the closest point to the vehicle path from the pedestrian
 - i. Pedestrians will only move perpendicular to the vehicle for this project, so the closest point to the vehicle path will hold true as the only possible collision point
 - ii. The vehicle path is defined as the area extending horizontally from the center of the vehicle and 1m in the +y and -y directions, to account for a vehicle width of 2m
 - b. Compare time the pedestrian will take to reach the collision point to the time the vehicle will take to reach the same point
 - i. Pedestrian distance from the collision point is calculated using the current pedestrian position from the stereo camera sensor output, 0.25m will be subtracted from this distance to account for the defined size of the pedestrian (circle with 0.5m diameter)
 - ii. Amount of time the pedestrian will take to reach the collision point will be calculated using the pedestrian distance from the collision point and the worst-case scenario assumption described above for pedestrian velocity
 - iii. Amount of time the vehicle will take to reach the collision point is calculated using the current vehicle position, current vehicle speed, and maximum vehicle acceleration which is defined as 0.25g (2.45m/s²)
 - c. A Pedestrian Hazard will be found when the pedestrian can reach the point before the vehicle will pass it, based on the times calculated above

4. Account for ALL pedestrians identified by camera sensor in this manner, to find all possible collisions
5. Activate braking system to respond to and avoid the closest Pedestrian Hazard
 - a. Request enough deceleration from the brake-by-wire system, so that the vehicle will be able to stop before reaching the possible collision point from above
 - i. Distance needed to stop for this requirement is determined by current vehicle speed and maximum deceleration, which has been defined as 0.7g (6.87m/s²)
 - ii. Deceleration needed to keep the distance needed to stop less than the distance to the collision point will be calculated by the algorithm, using the current vehicle speed as well as the distance to the possible collision point
 - iii. The algorithm will account for a 200ms delay time to reach the requested deceleration amount by decreasing the distance to the possible collision point by the current vehicle speed multiplied by 200ms
 - iv. The output of the algorithm will be the minimum deceleration needed to be requested to avoid a collision, this value will be sent as an input to the brake-by-wire system
 - v. A brake force inaccuracy of +/- 2% will also be considered by the algorithm when calculating brake force needed, brake force requested from the brake-by-wire system will be increased by this amount to mitigate the effect of the inaccuracy

Efficiency: Added time to trip resulting from collision avoidance should be minimized

The functional requirements for efficiency and its related constraints are outlined below:

1. The algorithm described above will output a deceleration amount that is not more than necessary to avoid the collision, to ensure time efficiency
2. An accurate and precise calculation of the minimum deceleration needed for the vehicle to avoid a possible collision is critical for this requirement
3. This requires the logic behind the algorithm to be effective, without introducing unnecessary margins for error
4. The added time will be calculated as the difference between the time the vehicle reaches its destination with the system off (collisions not avoided) and the time the vehicle takes with the system turned on (all collisions avoided)
5. To reduce added trip time, maximum acceleration should be applied when a possible collision is not detected, until the vehicle reaches the steady state speed of 50kph (13.9 m/s)

Additional Requirements and Constraints:

Fail Safe Mode: Implement an operation mode that avoids all collisions while response time for braking is significantly increased

The functional requirements for the Fail Safe Mode and its related constraints are outlined below:

1. The functional requirements for efficiency and its related constraints are outlined below:
2. Under Fail Safe Mode the response time to reach requested deceleration amount by the brake-by-wire system will be increased from 200ms to 900ms
3. When operating in this mode, this increased value will need to be used by the algorithm in calculating the minimum deceleration needed to avoid collisions

Security Requirements:

1. Implementation of a protocol to authenticate incoming sensor data towards the controller.
 - a. Private keys in each end-point with additional configured keys in order to encrypt all data outgoing from sensors and decrypt coming into the controller.
 - i. This could be a good idea to implement in all car sensors that send packets to the controller to minimize the possibility of intercepted vehicle information.
 - b. Checksums in place at the controller to ensure the data being received is complete and void of error.
 - i. This is a good measure to take to ensure the data being acted upon is correct. The last thing we would want is some bits being flipped and the distance to impact potentially skewed.
 - c. Maximum values assigned for steady state velocity – ensuring that even if somehow the velocity of the car exceeded this steady state velocity, the car would not continue to accelerate.
 - d. Add redundancy to the Brake by Wire System by ensuring a message would not be faked to the controller – indicating a brake request was received for example.
 - i. This could be in the form of a protocol discussed above.
 - ii. Ensure multiple checks to the Boolean field indicating active brakes.
 - iii. If a bad request is sent, make sure the controller has a way to resend and ensure the brakes respond as intended. If not, lock them up; we do not want the passengers or pedestrians at risk if the system is not responding.
 - e. While implementing this system, it is safer to use good design principles as discussed in class. In particular, minimizing coupling and maximizing cohesion play a large role in security. These also lead to good modularity and encapsulation -- making it easier to diagnose and maintain in the future.
 - i. Less ways for example to access the brake system means a more controlled entry point through which all communication exists.
 - ii. Maximizing the cohesion of operations within one system is vital to precisely controlling how that system operates. In a high assurance system such as this, we want all brake operations for example to be handled by one system – minimizing the communication errors that might exist otherwise.

4 Modeling Requirements

Domain Model:

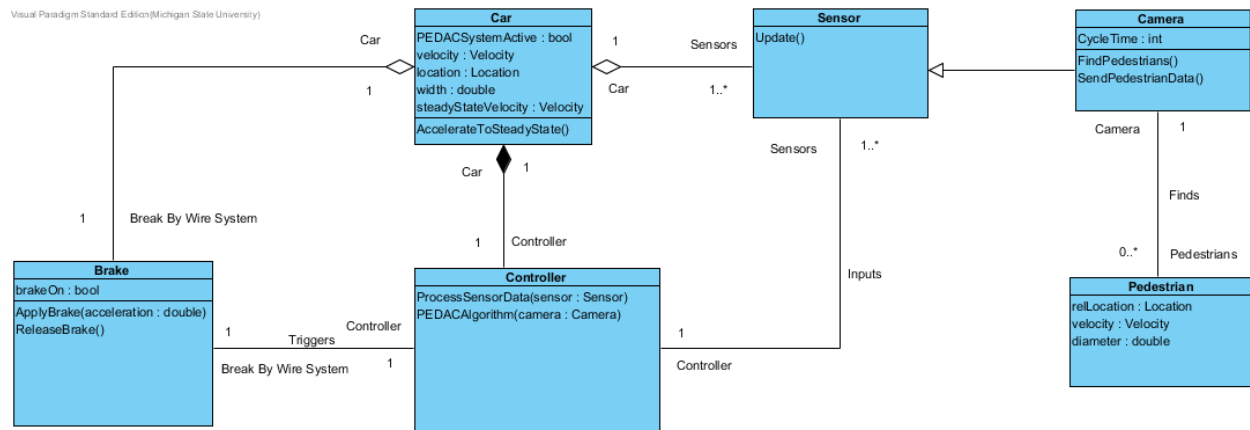


Figure 4.1

Car: Car object includes an indicator to whether the system is on (PDACSystemActive), as well as a velocity vector, a location in space, the width of the car, and the steady state velocity. A call to `AccelerateToSteadyState()` will bring the velocity vector back up to the steady state velocity – it cannot be greater than this.

Sensor: This is the abstract base class for the sensors on the car. Every sensor on the car gets a call to update every 1ms – for the camera, this is where Cycle Time would be incremented.

Camera: The stereo camera sensor on this car is used to find pedestrians and send the pedestrian data to the controller every 100ms; The Cycle Time timer is used to determine when this is reached. Finding pedestrians and sending data does not happen on the sensor update call.

Pedestrian: Pedestrians are identified and their data sent to the controller of the car. They have a relative location to the car, a velocity vector, and a diameter indicating a collision zone.

Controller: The controller object processes all sensor data; When this data is from the camera and there are pedestrians, the controller uses the `PEDACAlgorithm()` to determine proper actions to be taken.

Brake: This is the Brake by Wire System used to issue autonomous brake requests including applying brake by some acceleration and releasing the brake – indicating to the car that it is safe to resume its steady state speed.

Data Dictionaries:

Element Name		Description
Brake		This class represents the Brake by Wire System that the Pedestrian Collision Detection System communicates with. It controls the cars braking capability – applying brakes or releasing brakes.
Attributes		
	brakeOn : bool	A Boolean value representing the brakes being applied. True reflects the brakes being active and False reflects inactivity.
Operations		
	ApplyBrake(acceleration : double)	This method applies acceleration to the brake system. The acceleration's direction is in the opposite direction of the steady state velocity.
	ReleaseBrake()	This operation releases all the brakes within the system – allowing the car to resume a steady state velocity.
Relationships	The Brake by Wire system helps make the car what it is and communicates with the car's controller. The controller triggers the brake system and the brake system sends responses in the form of Boolean manipulation.	
UML Extensions	None.	

Element Name		Description
Camera		This is the primary sensor in which we get our pedestrian data from. This includes the relative location of each pedestrian with respect to the car and their velocity.
Attributes		
	CycleTime : Int	This is the timer intended to make sure packets of data are sent to the controller at the correct times.
Operations		
	FindPedestrians()	This operation finds any pedestrians in view of the camera and instantiates Pedestrian objects in which their data can be passed to the controller.
	SendPedestrianData()	This operation is called every 100ms and finds any pedestrian data to start. Then, if there is data to send, the controller is passed this sensor object in the processing of this data.
Relationships	The camera class is derived from the Sensor base class and creates Pedestrian objects upon calling SendPedestrianData() every 100ms.	
UML Extensions	None.	

Element Name		Description
Car		This is the main object of manipulation for the system. This is the space object in which the pedestrians collide with and the direct recipient of action placed by the PEDAC Algorithm.
Attributes		
	PEDACSystemActive : bool	Boolean value indicating whether the system is active or not.
	velocity : Velocity	Vector indicating the speed in both the x and y directions.
	location : Location	Location coordinates in the x and y directions in meters.
	width : Double	The width of the car determines the impact zone the pedestrian could potentially collide with.
	steadyStateVelocity : Velocity	This velocity vector indicates the maximum velocity the system is capable of accelerating to. It is set upon starting the system.
Operations		
	AccelerateToSteadyState()	This operation is responsible for accelerating the car back up to the steady state velocity at the maximum positive acceleration.
Relationships	The controller is a composite component of the car as well as is the sensors and the brake system. However, the controller is needed for the car to be classified as a car.	
UML Extensions	None.	

Element Name		Description
Controller		The controller is the main 'brain' behind the car. This is where the PEDAC Algorithm is applied and where all requests for actions to be taken by the car originate. Also, the controller is responsible for processing the sensor data.
Attributes		
	None.	None.
Operations		
	ProcessSensorData(sensor : Sensor)	This operation is called from each sensor when operations might need to take place. The sensor passes itself to the controller – therefore giving its data to specific algorithms within the controller.
	PEDACAlgorithm(camera : Camera)	If pedestrians are found by the camera, this method is called by the sensor for any pedestrian data to be recognized by our system -- providing the correct responses to the brake by wire system.
Relationships	The controller is a composite element of the car and communicates with sensors and the brake by wire system. Sensor data functions as the input and this then triggers responses in the brake system.	
UML Extensions	None.	

Element Name		Description
Pedestrian		This class represents a pedestrian object created by the camera when finding and communicating pedestrian locations and velocities to the controller.
Attributes		
	relLocation	The relative location of the pedestrian with respect to the car. This has both x and y components in meters.
	velocity	This is the velocity of the pedestrian. It has both x and y components in meters per second.
	diameter	This is the diameter of the pedestrian. This info is used to calculate potential collisions.
Operations		
	None.	None.
Relationships	This class is instantiated by the camera object upon processing data every 100ms. The camera finds these pedestrians in view of the vehicle for the controller to interpret.	
UML Extensions	None.	

Element Name		Description
Sensor		This is the abstract base class for all sensors within the car. The camera extends this class and performs sending data upon meeting requirements in the update call.
Attributes		
	None.	None.
Operations		
	Update()	Called every 1ms. This is used by the camera to increment CycleTime and send data at the right times.
Relationships	Sensors are composite elements of the car and communicate directly with the controller. When messages are sent, everything goes to the controller.	
UML Extensions	None.	

Use-Case Diagram:

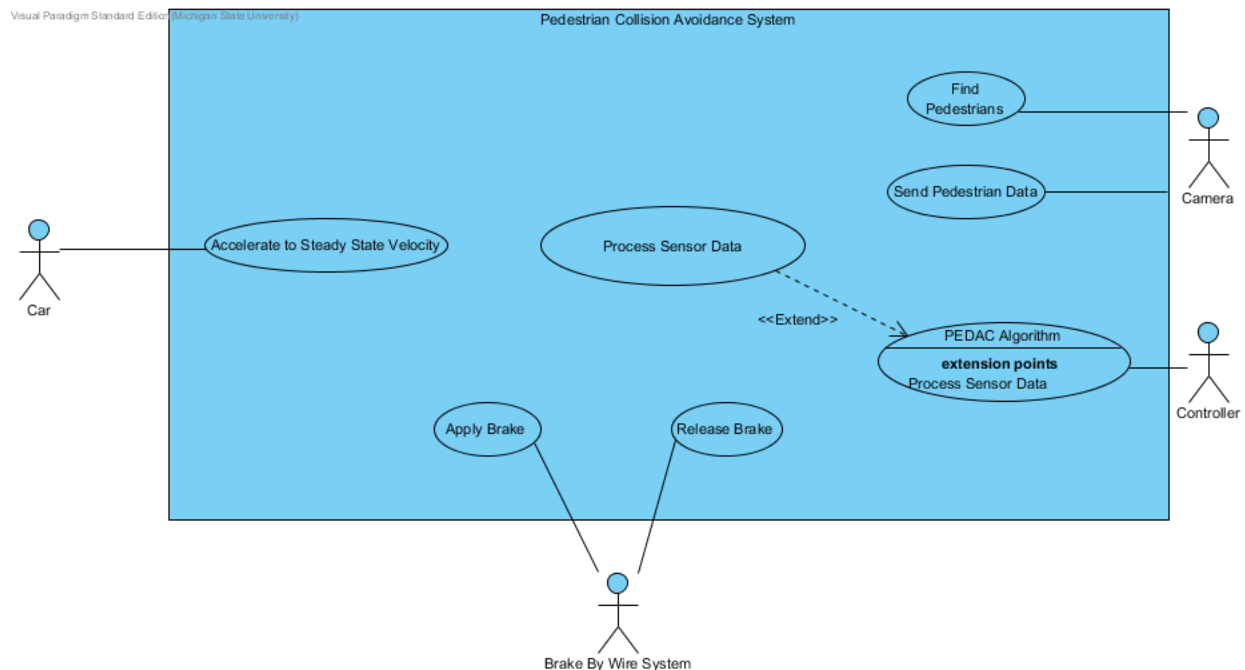


Figure 4.2

Use Case: Accelerate to Steady State Velocity

Actors: Car

Description: The car is accelerating up to its steady state velocity – its velocity cannot be beyond this.

Type: Primary

Use Case: Apply Brake

Actors: Brake by Wire System

Description: The Brake by Wire System is issued a request to apply the brake with acceleration – resulting in the car decelerating.

Type: Primary

Use Case: Release Brake

Actors: Brake by Wire System

Description: The Brake by Wire System is issued a request to release any brake that is on.

Type: Primary

Use Case: Find Pedestrians

Actors: Camera

Description: The camera finds the pedestrians in view and keeps track of their relative position to the car and pedestrian velocity.

Type: Primary

Use Case: Send Pedestrian Data

Actors: Camera

Description: The camera sends the pedestrian information in packets every 100ms to the controller.

Type: Primary

Use Case: PEDAC Algorithm

Actors: Controller

Extends: Process Sensor Data

Description: The controller, upon receiving information from the sensor data, processes the data. If the data is from the camera and pedestrians are present, the PEDAC Algorithm is called within this processing operation. Here, the controller decides proper action for the car based on pedestrian locations relative to the car and their velocities.

Type: Primary

State Diagrams:

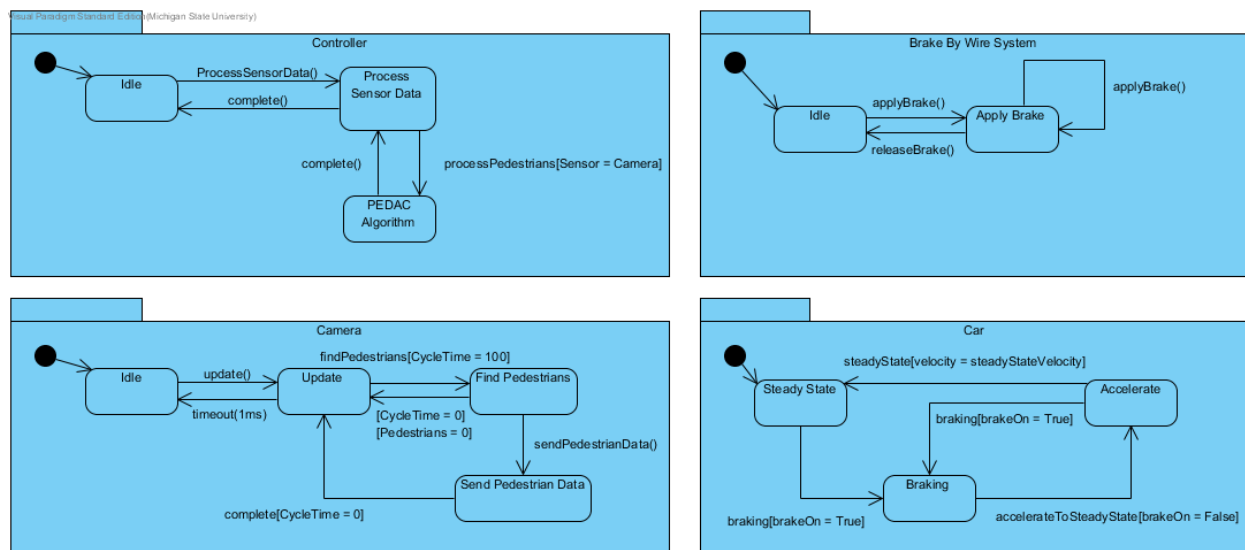


Figure 4.3

Controller:

The controller starts and stays in the idle state until ProcessSensorData() is called – indicating some action may be required. If the Camera initiated this request, the controller enters the PEDAC Algorithm state; here, the camera data is processed and actions are given to the Brake by Wire System if necessary until completion – where it once again enters the idle state. If it is not the Camera who initiated the request, the processing continues until completion and eventually ends back in the idle state.

Brake by Wire System:

This system starts in the idle state until a request to apply the brakes is given – where it then enters the Apply Brake state. If subsequent requests to apply brake are given, the system stays in the Apply Brake state with a potentially different value for the acceleration in which to brake by. If the system is in the Apply Brake state and a request to release the brakes is given by the controller, the system goes back into the idle state.

Camera:

The camera stays in the idle state until a request to update – in which it moves to the Update state. If the Cycle Time is 100ms, the camera moves into the Find Pedestrians state; if not, Cycle Time is incremented by 1ms and a timeout occurs for 1ms – where after, the camera will be idle again. From the Find Pedestrian state, if pedestrians are found, the camera sends the pedestrian data to the controller; Otherwise, Cycle Time is set to 0 and the camera returns to the Update state. If the camera is in the Send Pedestrian Data state, when complete, Cycle Time returns to 0 and the camera is moved into the rest of the Update state.

Car:

The car starts in the steady state. If a brake request is issued, the car goes into a braking state until brakeOn is set to false – where the car then moves back into the Accelerate State before going back to the Steady State. If while in the acceleration state the car is now braking, the car will go back into the braking state until they are released.

Sequence Diagrams:

Scenario 1 and 2 (Figure 3.1):

In these scenarios, the pedestrian is moving and then stops right in front of the car. The controller processes camera data without action until the pedestrian is seen and is too close. Then, while processing the camera data again (in the PEDAC Algorithm), the controller issues a brake request to the Brake by Wire System. This happens repeatedly until the car is stopped right in front of the pedestrian – free of collision.

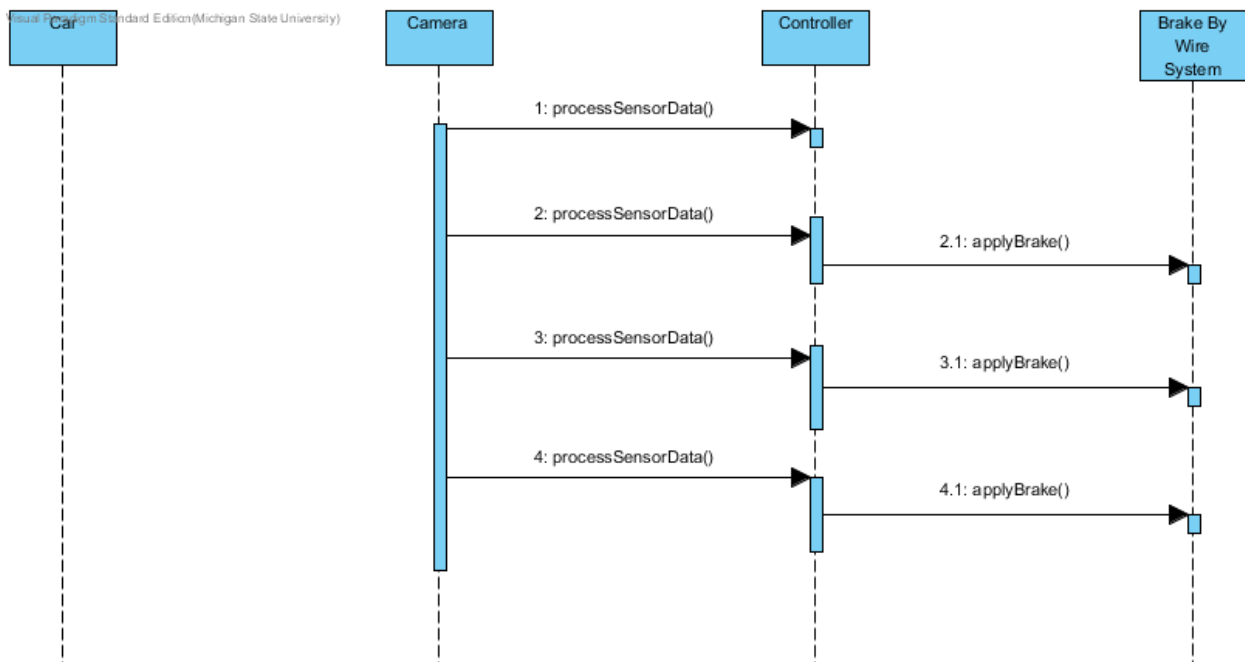


Figure 4.4

Scenario 3 and 4 (Figure 3.1):

In these scenarios, the pedestrian is moving and then stops in time – where the car will not collide with the pedestrian. The controller repeatedly processes camera data every 100ms until the pedestrian in sight becomes too close. Anticipating the worst, the controller issues a brake request in the next data processing to make sure that if the pedestrian were to start moving again, the car would have enough time to stop. When it is clear that the car will pass the pedestrian without collision, the controller tells the Brake by Wire System to release all brakes – where the system responds to the controller with conformation. The controller then requests for the car to accelerate to the steady state speed at the maximum acceleration. One more processing request comes from the camera to the controller; however, no action is needed as the car will not come close to hitting the pedestrian.

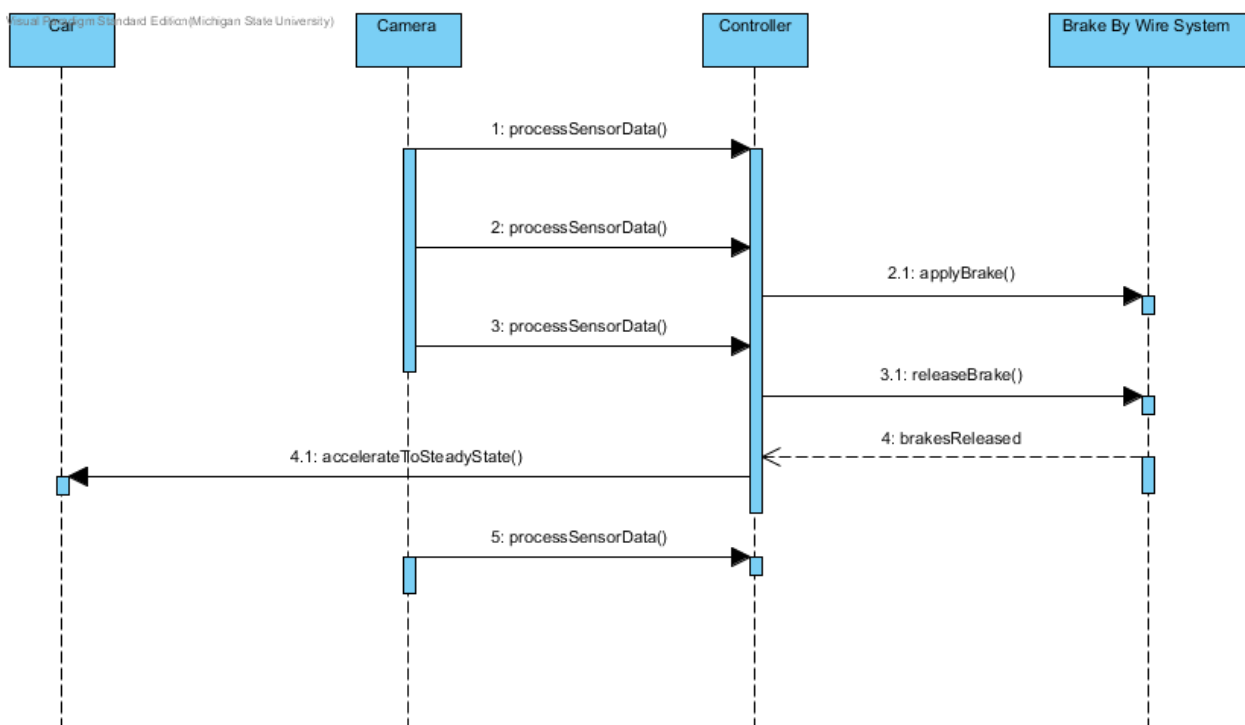


Figure 4.5

Scenario 5, 6, and 7 (Figure 3.1):

In these scenarios, the pedestrian remains static in front of the car and then starts to move. Repeated requests for the controller to process incoming camera data happen before the pedestrian is too close to continue at steady state velocity. On the next request to process camera data, the controller issues a request for the Brake by Wire System to apply the brakes. This happens until the pedestrian starts to move out of the way of the car – where the controller then issues a request for the Brake by Wire System to release the brakes. Upon conformation from the Brake by Wire System, the controller issues a request for the car to accelerate back up to the steady state velocity – passing the pedestrian safely.

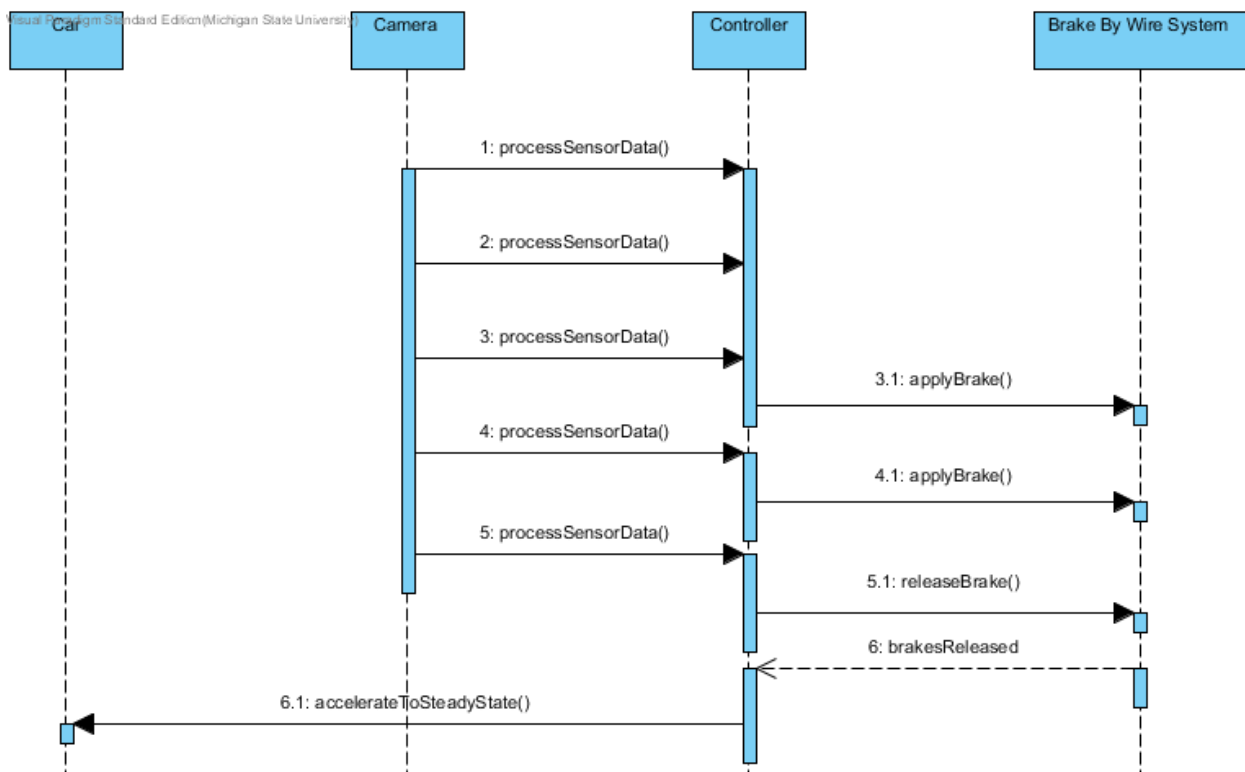


Figure 4.6

Scenario 8 and 9 (Figure 3.1):

In these scenarios, the pedestrian remains static in the way of the car. Repeated requests to process the camera data are sent from the camera to the controller until the pedestrian is too close – where then a request also takes place from the controller to the brake by wire system to apply the brakes. This is repeated until the car is stopped in front of the pedestrian.

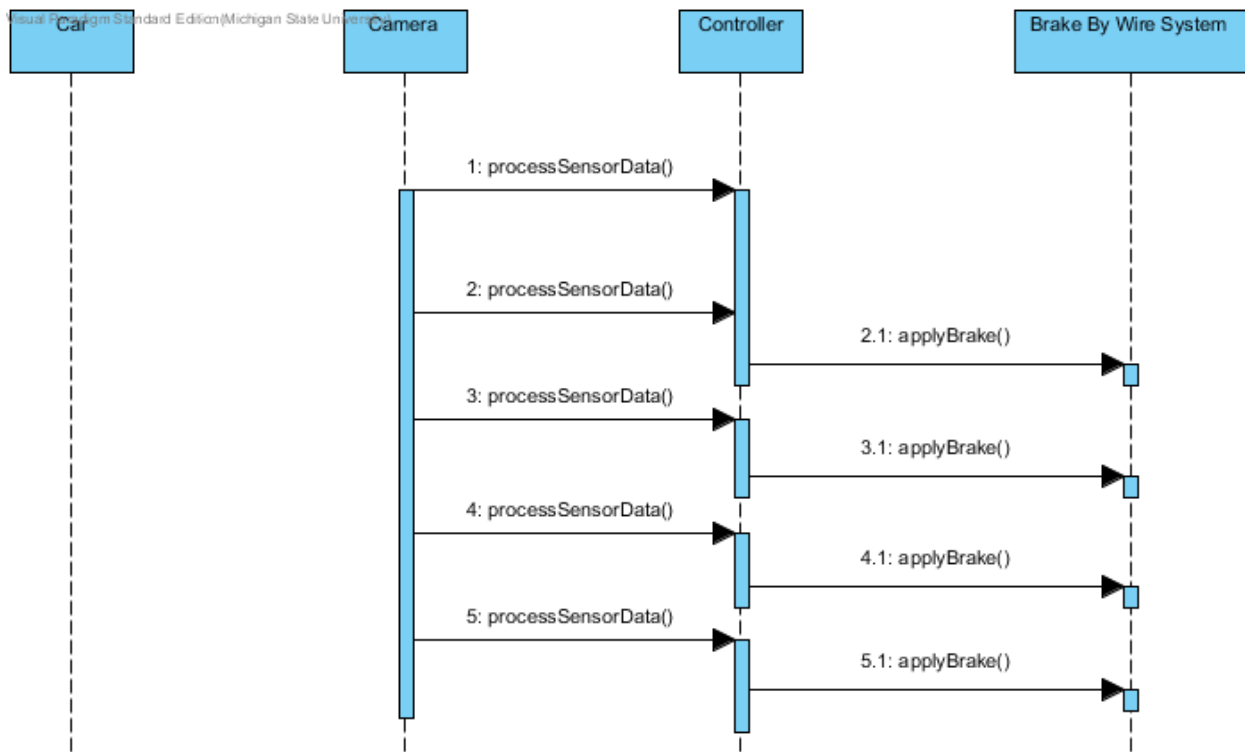


Figure 4.7

Scenario 10 (Figure 3.1):

In this scenario, the pedestrian remains static away from the car. The controller processes the camera data every 100ms until the pedestrian is deemed too close. The controller then issues a request to the brake by wire system to apply light brakes – this is determined by the PEDAC Algorithm. Eventually, when the pedestrian is at a safe distance from the car, the controller issues a request for the brake by wire system to release all brakes – resulting in the controller requesting the car return to its steady state velocity upon a confirmation message that the brakes are released. Subsequent calls to `processSensorData()` are made, however, there are no pedestrians in view.

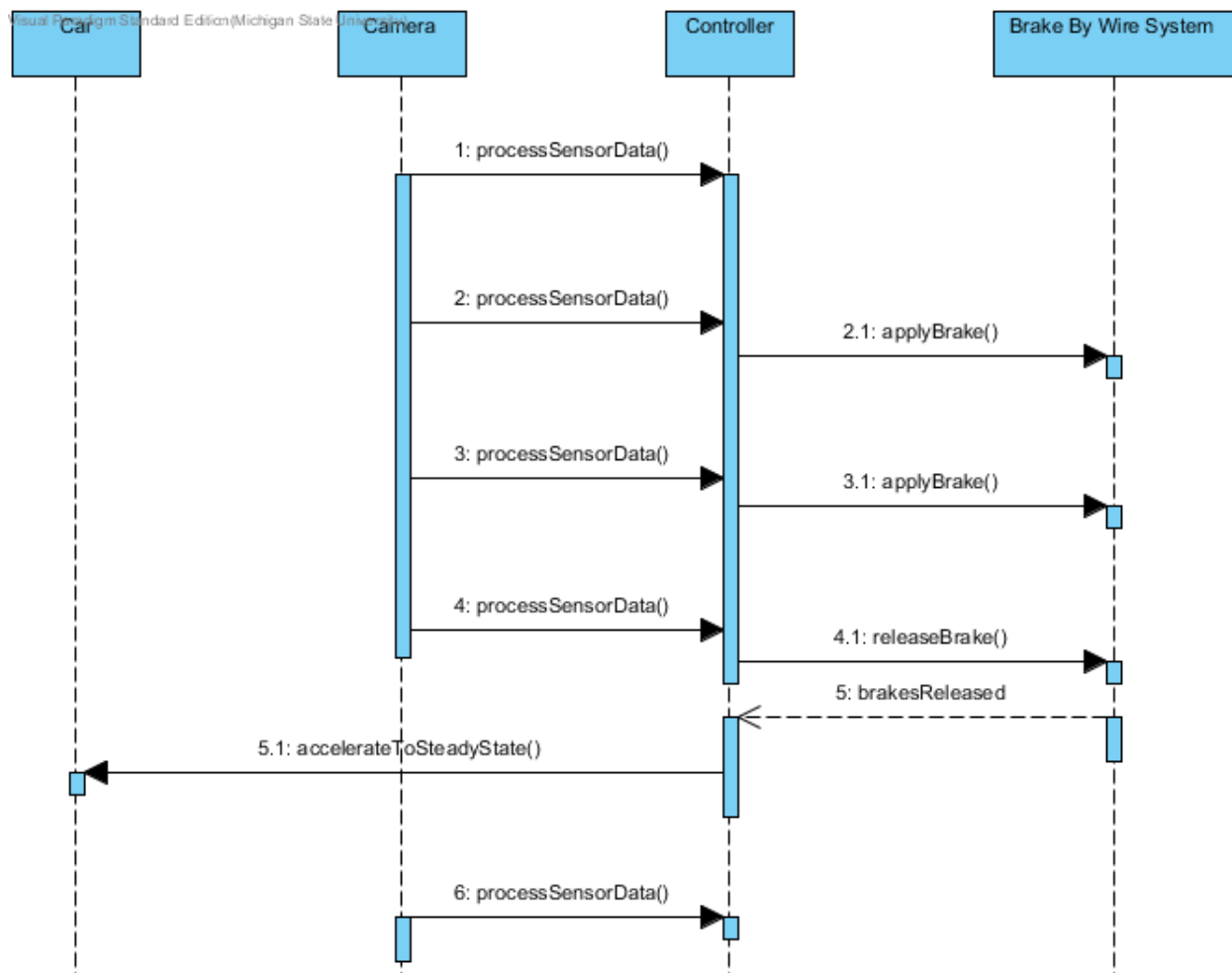


Figure 4.8

5 Prototype

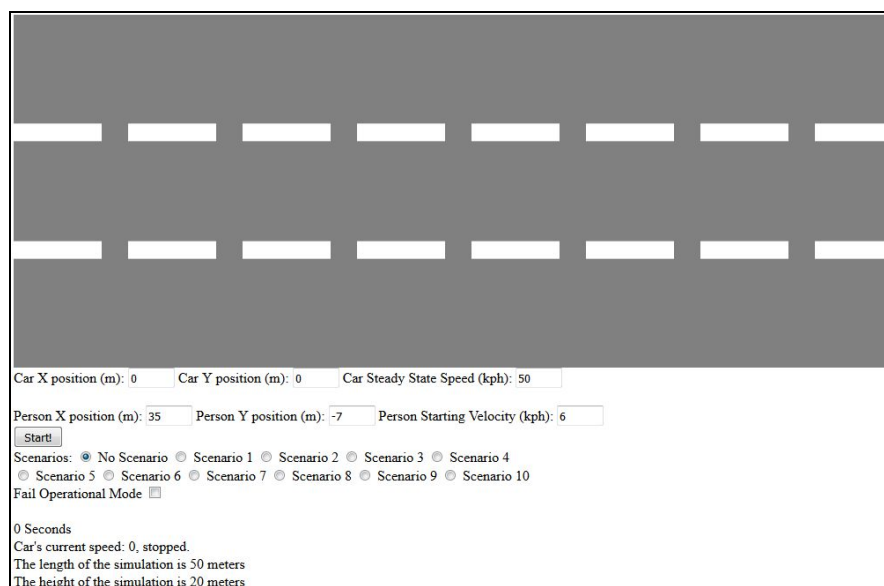
Our prototype will allow the user to set the starting position and velocity of a car and pedestrian and run a simulation with our algorithm to see if the car can avoid the collision. It will also allow the user to select from ten built-in scenarios and select whether or not fail safe mode should be enabled for the simulation.

5.1 How to Run Prototype

To run the prototype a web browser that has HTML5 and Javascript enabled is required. The prototype is available on the project website [2]. The user will input their desired starting parameters or press the button corresponding to the simulation they want to run and then press the “Start!” button. The simulation will run with realistic physics and is scaled properly. The simulation will run until the car and pedestrian collide (failure), the car reaches the end of the simulation area (success), or the car has been stopped for over one second without colliding with the vehicle (success) . Currently the simulation area is scaled to 50 meters long by 20 meters high. The size of the car and pedestrian only scales with the length of the simulation area, therefore the width (vertical dimension) will be distorted if changed from a 5/2 ratio. Additionally, the user has the option of enabling a “Fail Operational Mode” which increases the time it takes the car to start decelerating from 200 ms to 900 ms.

5.2 Sample Scenarios

Interface:



The interface is a web-based control panel for a simulation. It features a large gray rectangular area at the top representing the simulation space, with two horizontal dashed white lines indicating the initial positions of the car and pedestrian. Below this area are several input fields and controls:

- Car X position (m): 0
- Car Y position (m): 0
- Car Steady State Speed (kph): 50
- Person X position (m): 35
- Person Y position (m): -7
- Person Starting Velocity (kph): 6
- A "Start!" button
- A row of radio buttons for scenario selection: "No Scenario" (selected), "Scenario 1", "Scenario 2", "Scenario 3", "Scenario 4", "Scenario 5", "Scenario 6", "Scenario 7", "Scenario 8", "Scenario 9", and "Scenario 10".
- A checkbox for "Fail Operational Mode".
- Real-time status text: "0 Seconds", "Car's current speed: 0, stopped.", "The length of the simulation is 50 meters", and "The height of the simulation is 20 meters".

Figure 5.2.1

The interface the user sees immediately upon opening the prototype

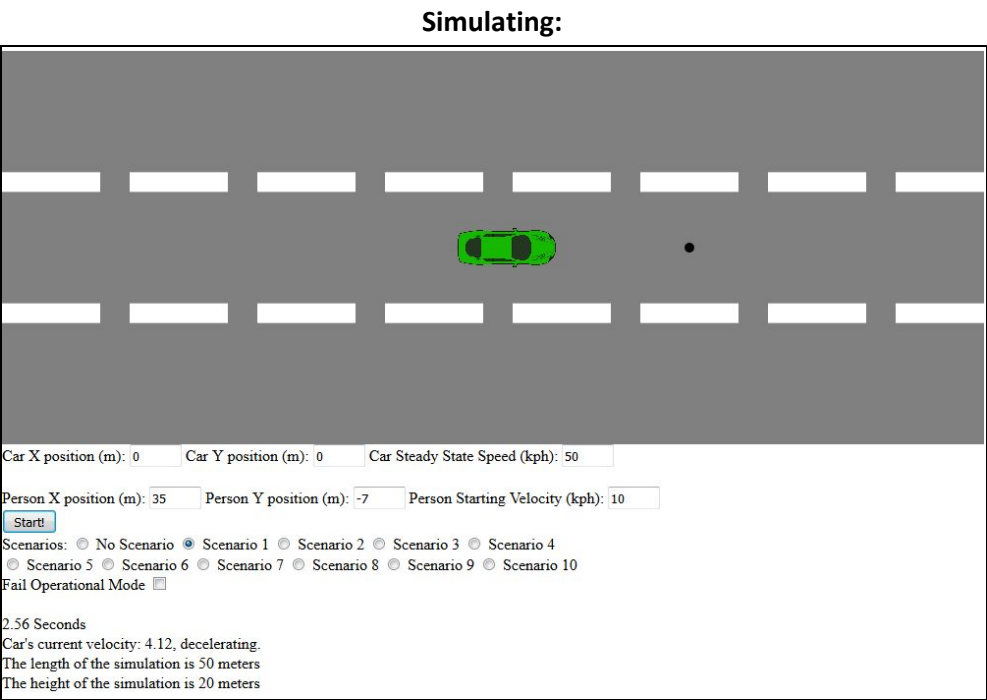


Figure 5.2.2

An example of a simulation. How long the simulation has been running as well as the car’s current velocity and whether or not it is accelerating or decelerating is shown toward the bottom.

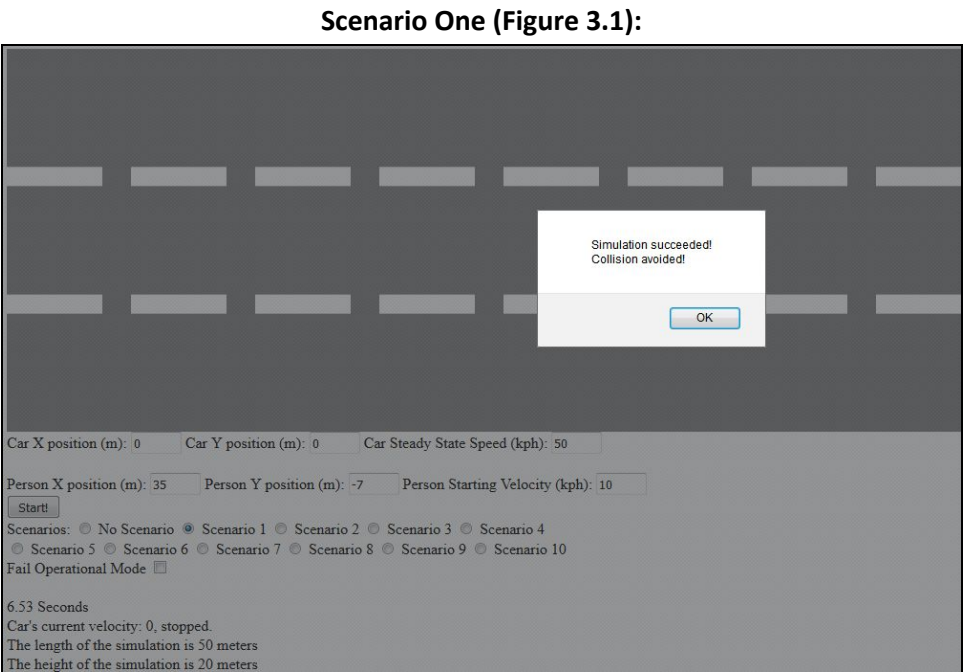


Figure 5.2.3

Scenario one succeeds by avoiding the collision.

Scenario Two (Figure 3.1):

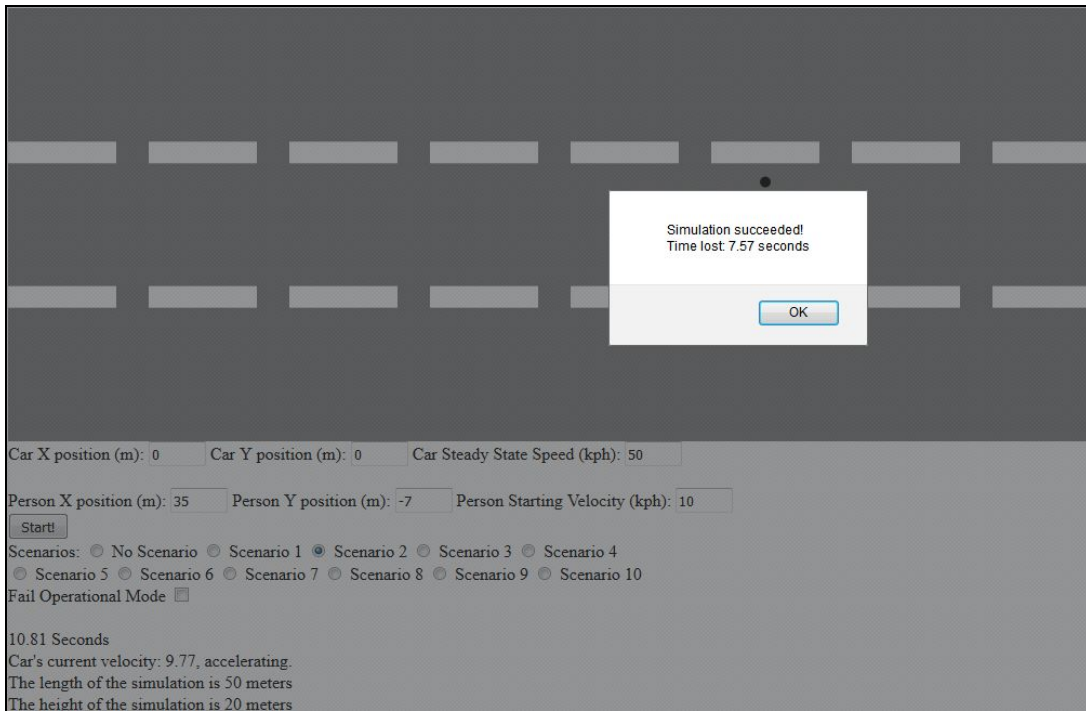


Figure 5.2.4

Scenario two succeeds with 7.57 seconds of lost time.

Scenario Three (Figure 3.1):

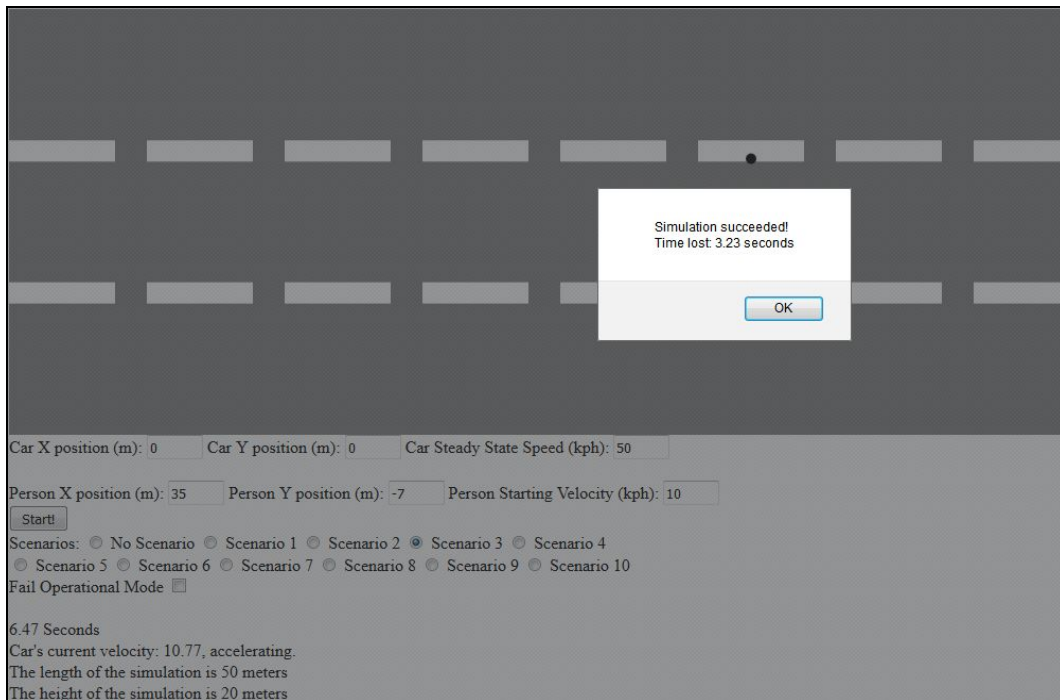


Figure 5.2.5

Scenario three succeeds with 3.23 seconds of lost time.

Scenario Four (Figure 3.1):

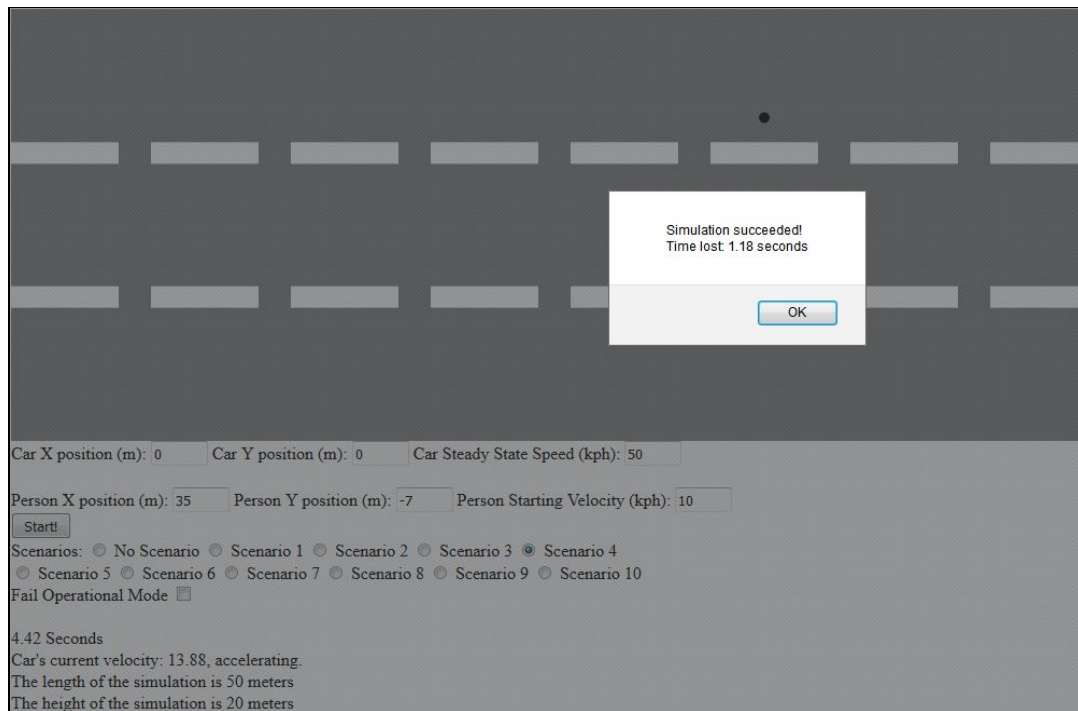


Figure 5.2.6

Scenario four succeeds with 1.18 seconds of lost time.

Scenario Five (Figure 3.1):

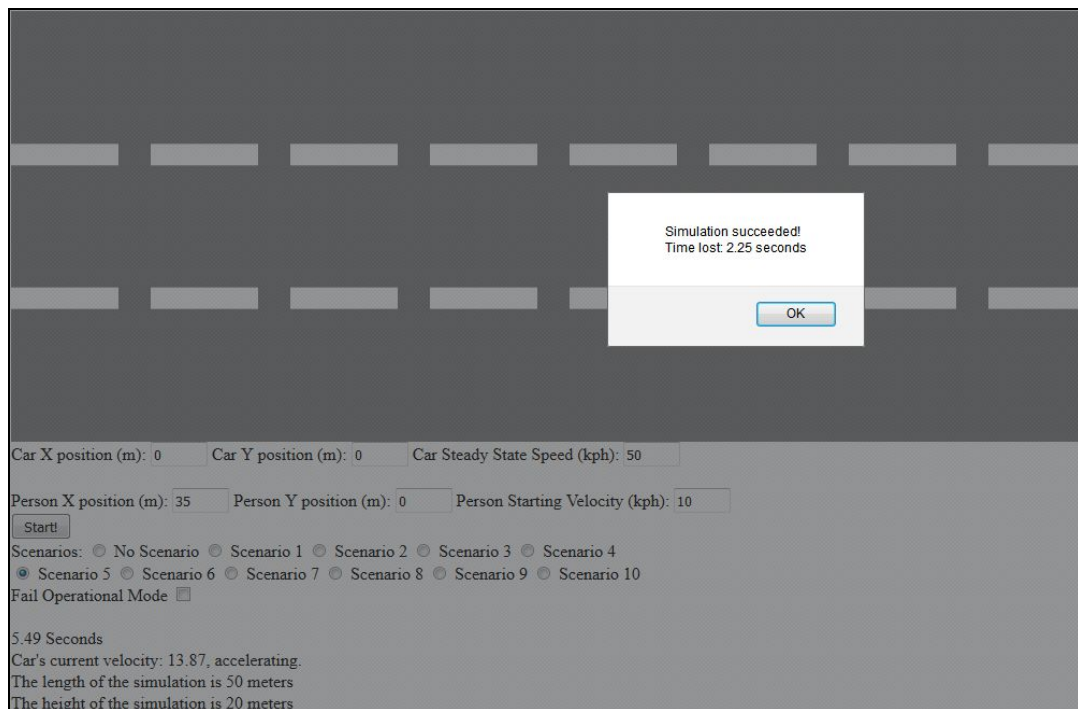


Figure 5.2.7

Scenario five succeeds with 2.25 seconds of lost time.

Scenario Six (Figure 3.1):

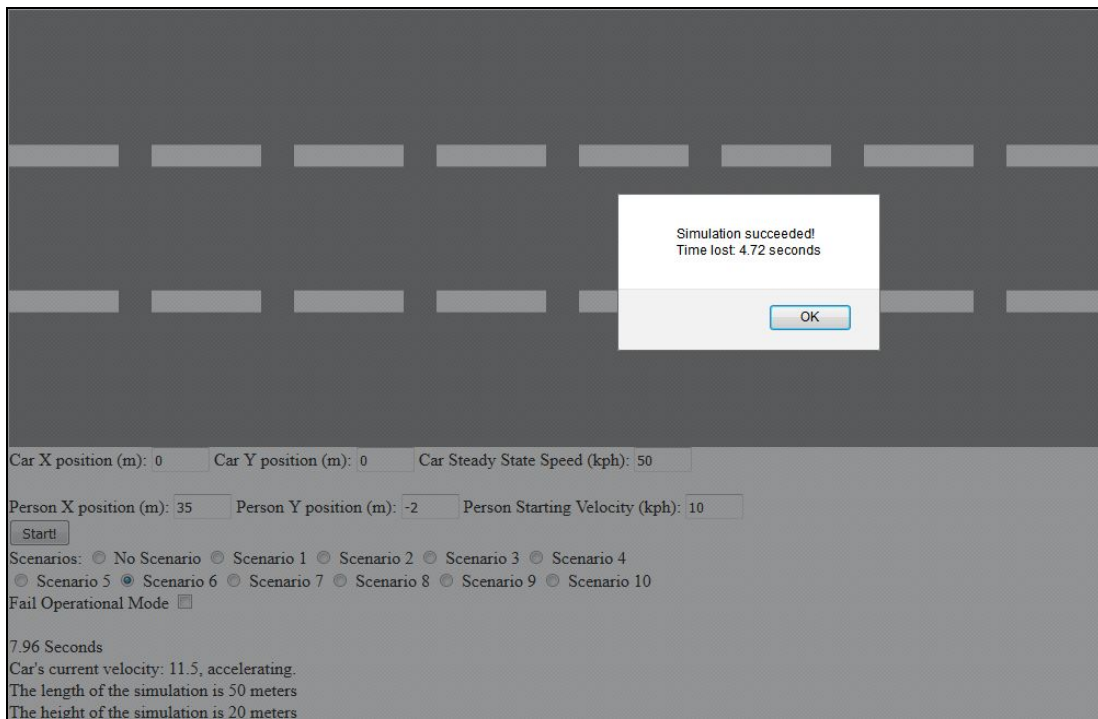


Figure 5.2.8

Scenario six succeeds with 4.72 seconds of lost time.

Scenario Seven (Figure 3.1):

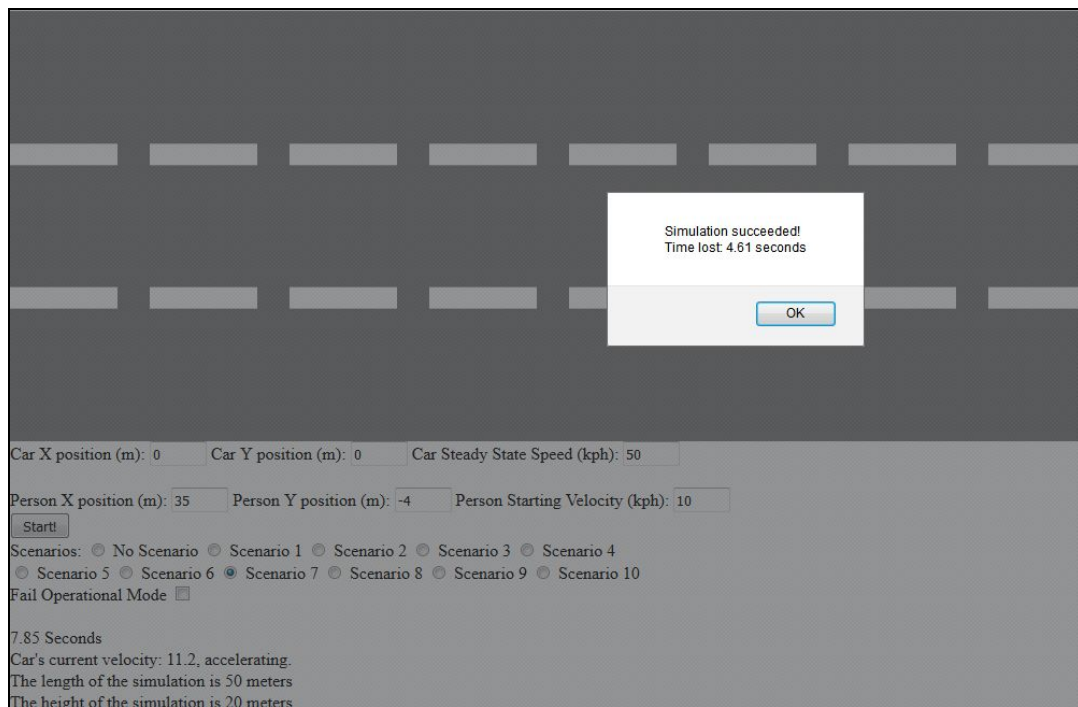


Figure 5.2.9

Scenario seven succeeds with 4.51 seconds of lost time.

Scenario Eight (Figure 3.1):

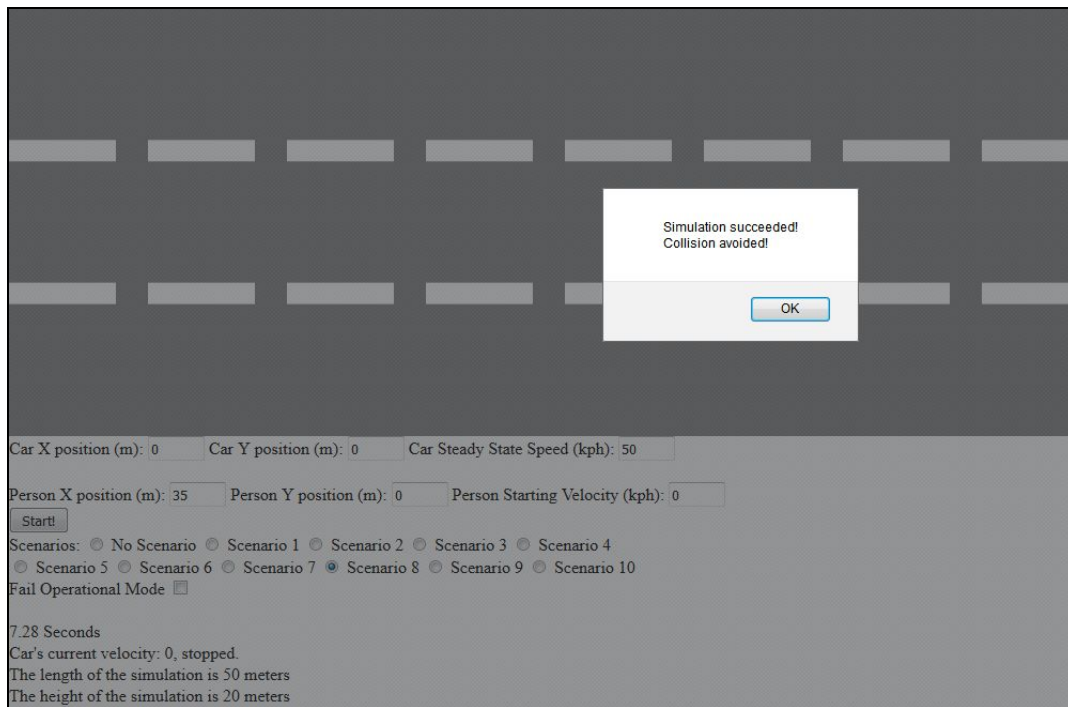


Figure 5.2.10

Scenario eight succeeds by avoiding the collision.

Scenario Nine (Figure 3.1):

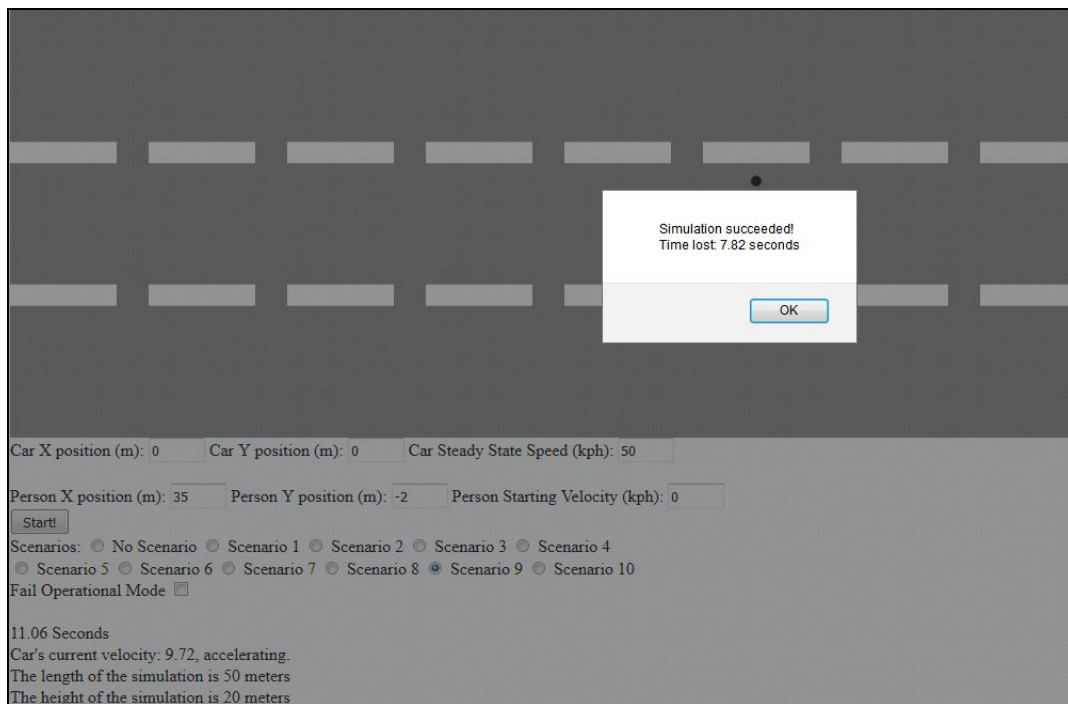


Figure 5.2.11

Scenario nine succeeds with 7.82 seconds of lost time.

Scenario Ten (Figure 3.1):

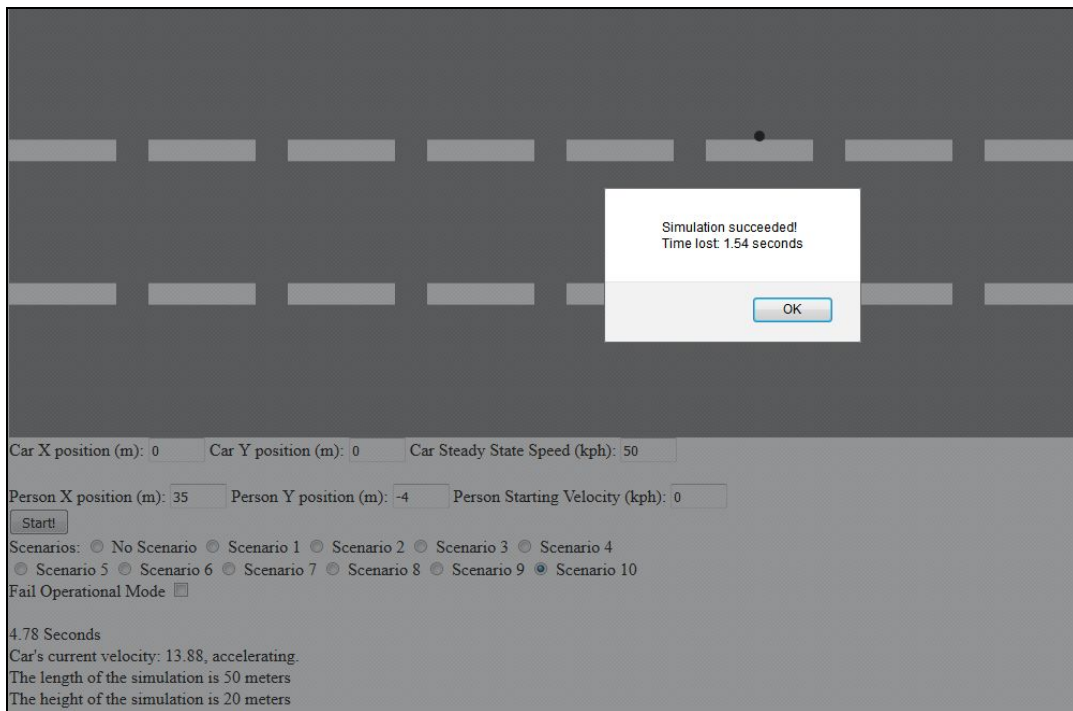


Figure 5.2.12

Scenario ten succeeds with 1.54 seconds of lost time.

Scenario One (Fail Mode Enabled):

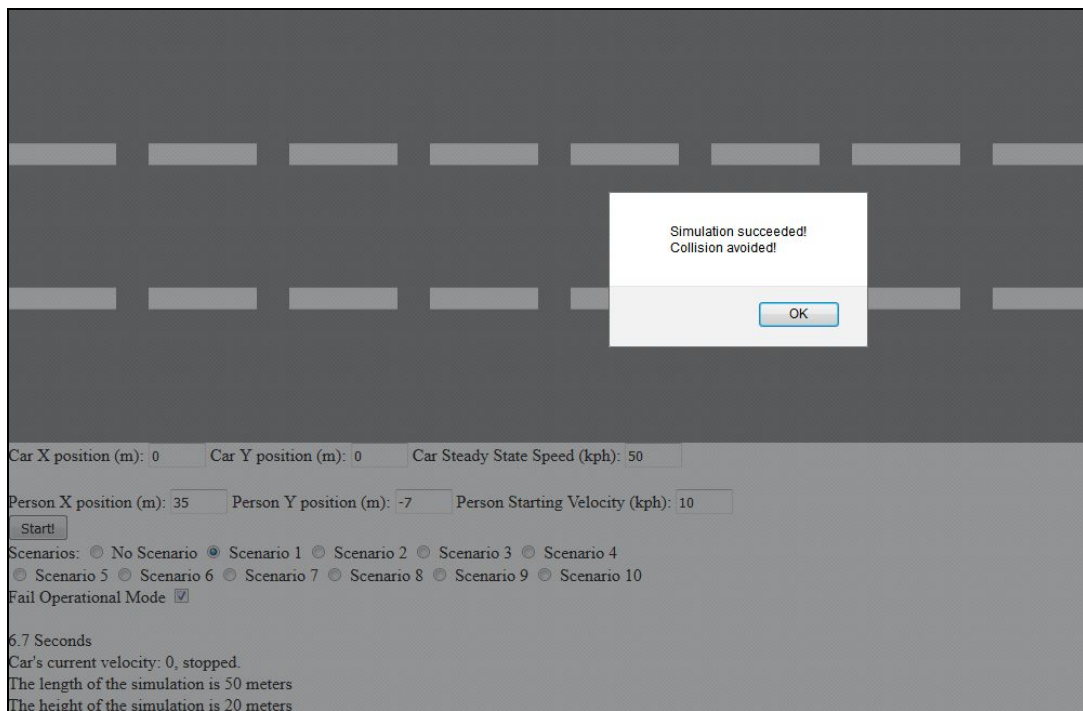


Figure 5.2.13

Scenario one succeeds by avoiding the collision when fail mode is enabled.

Scenario Two (Fail Mode Enabled):

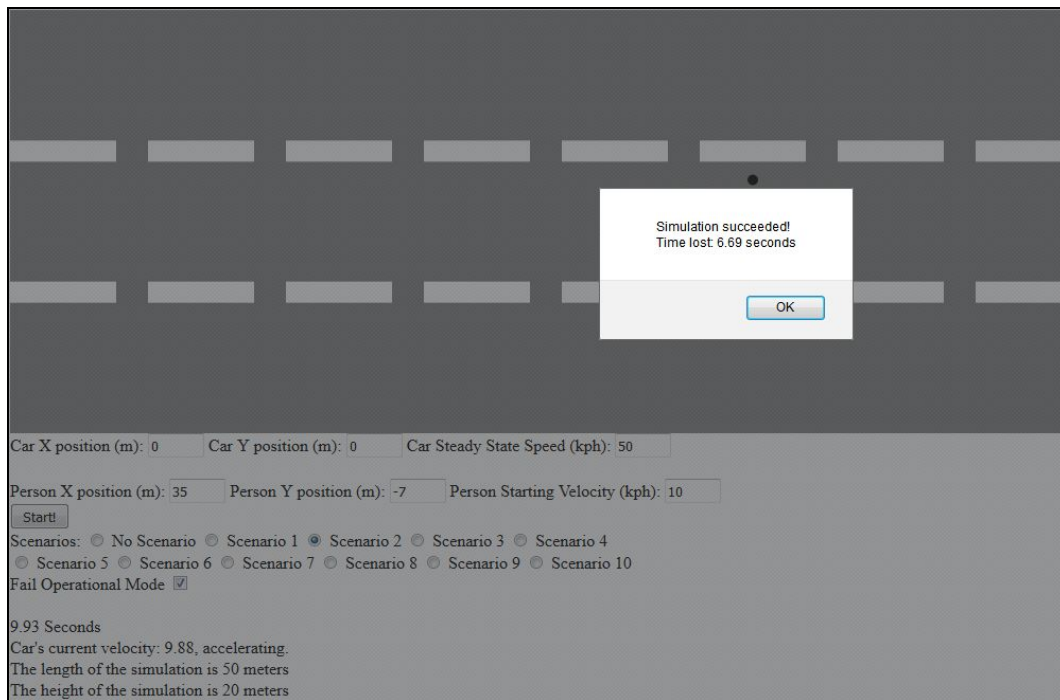


Figure 5.2.14

Scenario two succeeds with 6.69 seconds of lost time.

Scenario Three (Fail Mode Enabled):

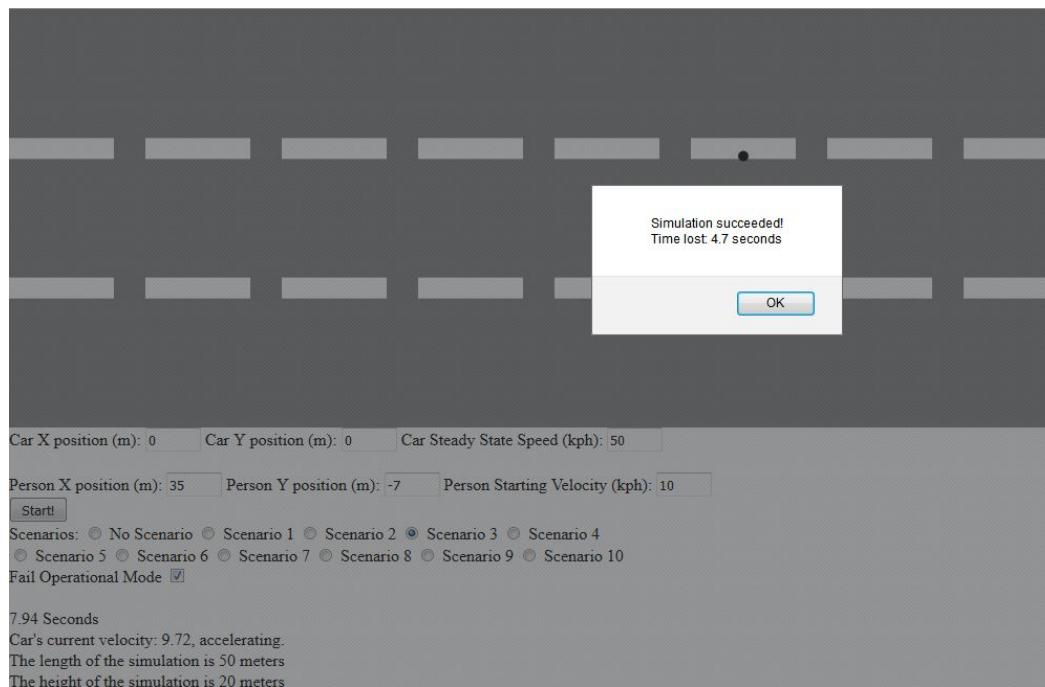


Figure 5.2.15

Scenario three succeeds with 4.7 seconds of lost time.

Scenario Four (Fail Mode Enabled):

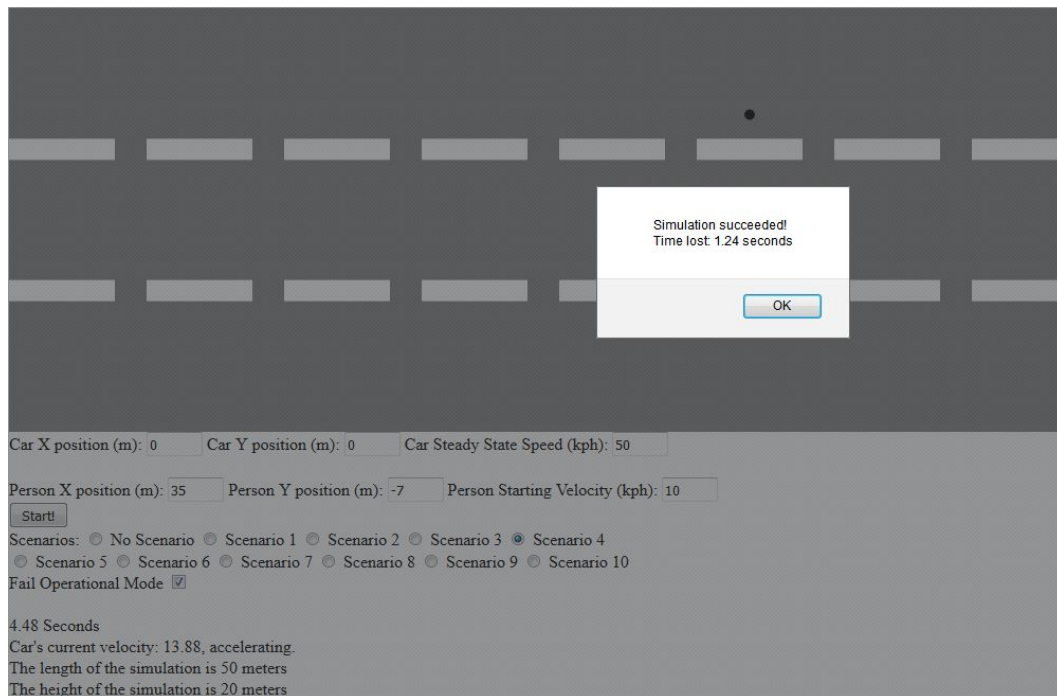


Figure 5.2.16

Scenario four succeeds with 1.24 seconds of lost time.

Scenario Five (Fail Mode Enabled):

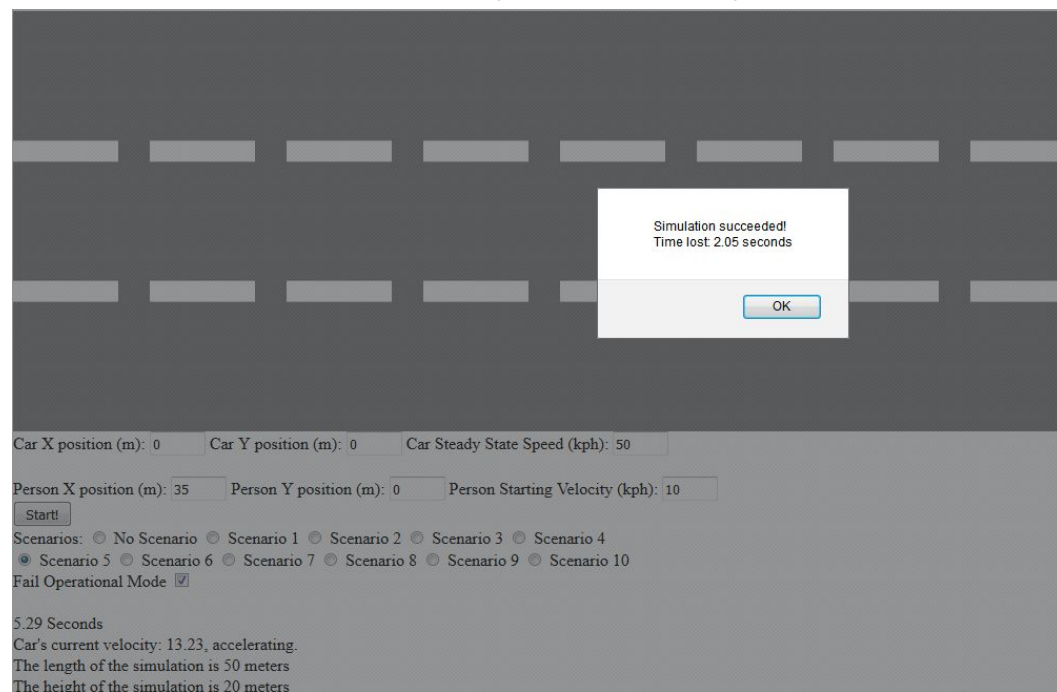


Figure 5.2.1

Scenario five succeeds with 2.05 seconds of lost time.

Scenario Six (Fail Mode Enabled):

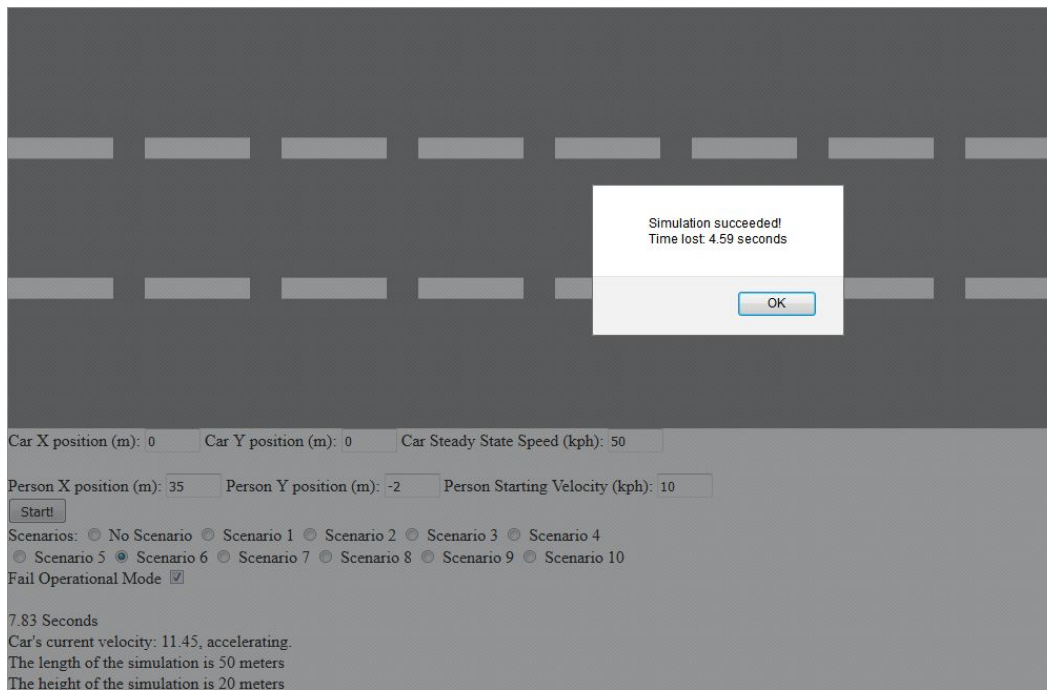


Figure 5.2.18

Scenario six succeeds with 4.59 seconds of lost time.

Scenario Seven (Fail Mode Enabled):

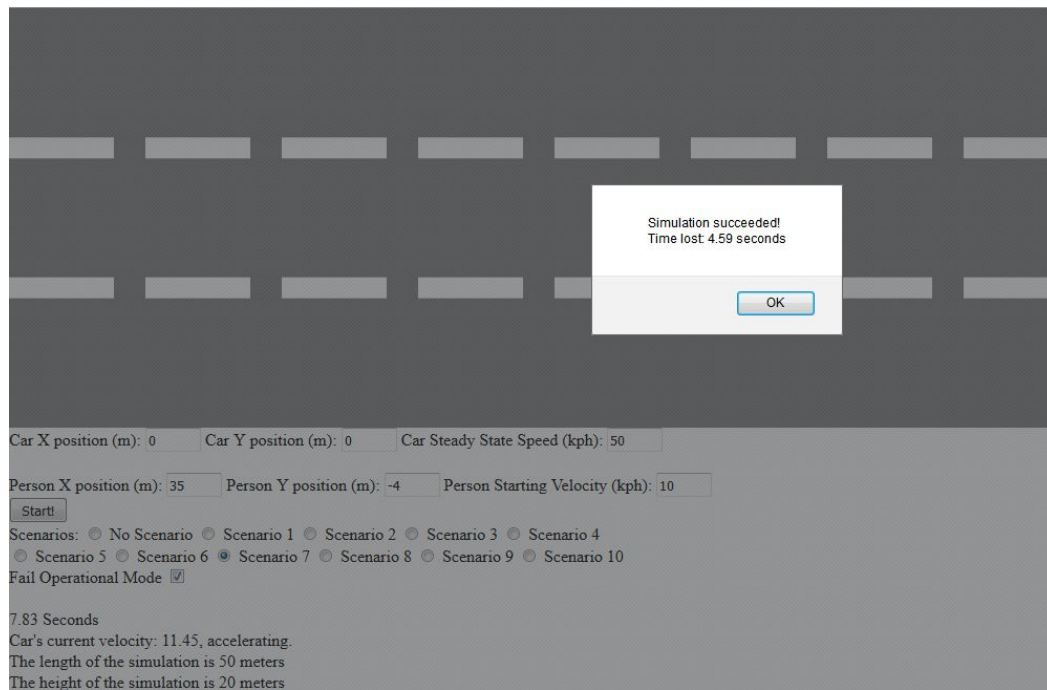


Figure 5.2.19

Scenario seven succeeds with 4.59 seconds of lost time.

Scenario Eight (Fail Mode Enabled):

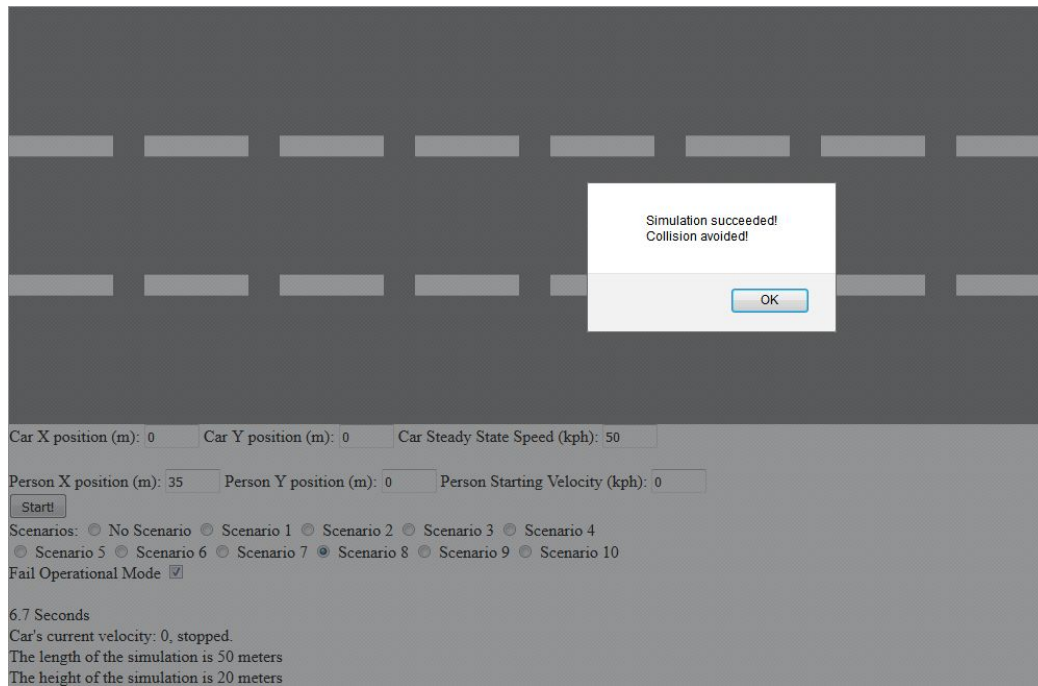


Figure 5.2.20

Scenario eight succeeds by avoiding the collision

Scenario Nine (Fail Mode Enabled):

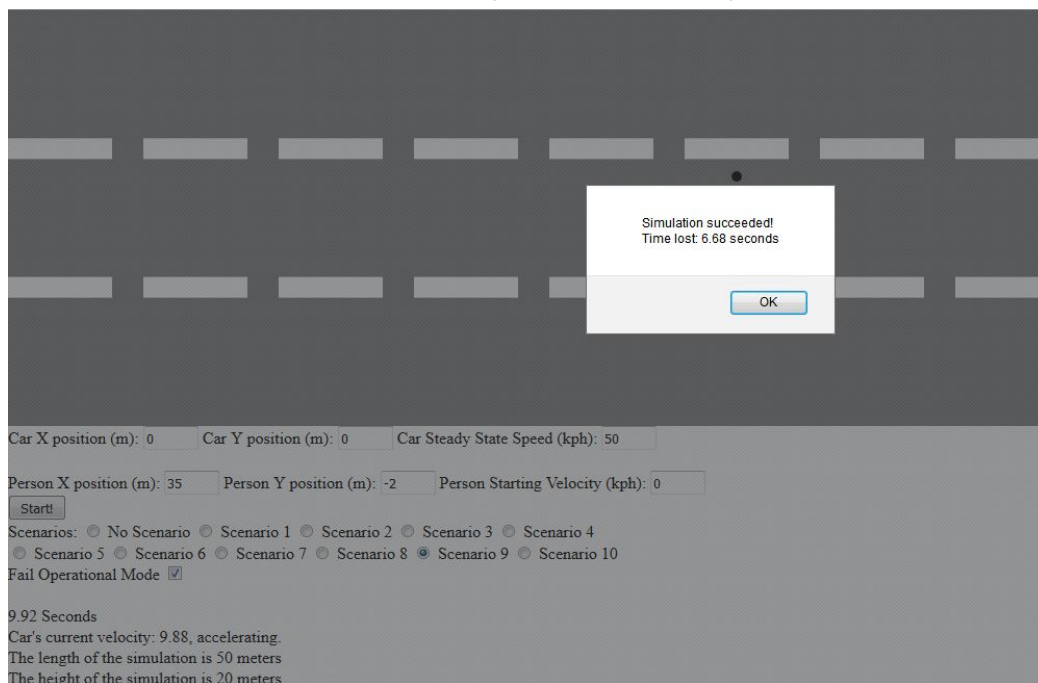


Figure 5.2.21

Scenario nine succeeds with 6.68 seconds of lost time.

Scenario Ten (Fail Mode Enabled):

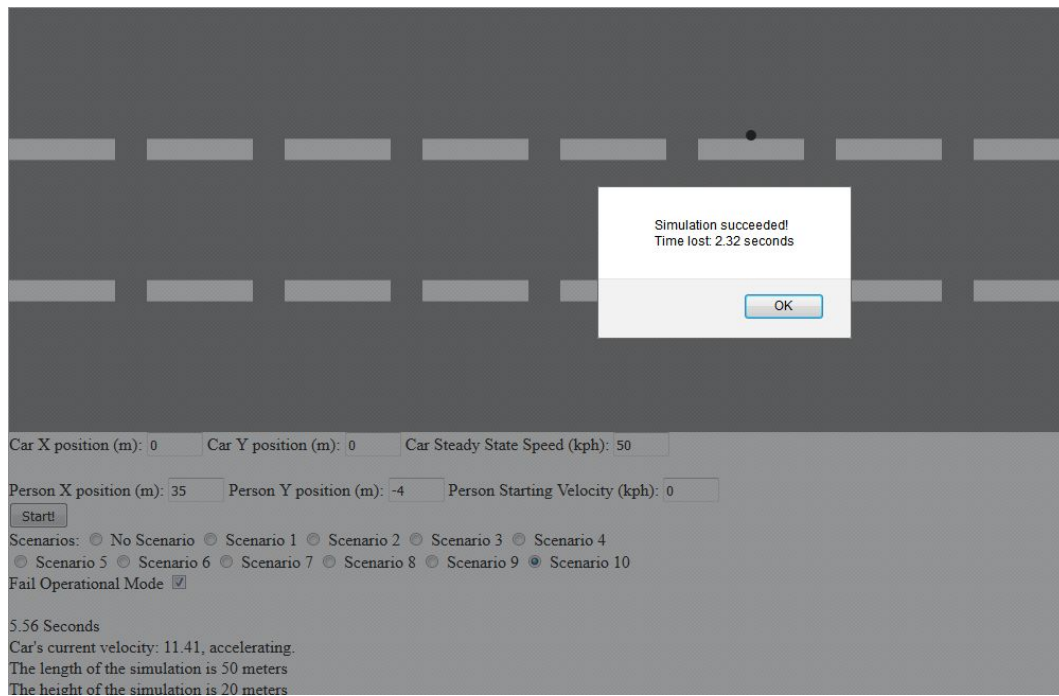


Figure 5.2.22

Scenario ten succeeds with 2.32 seconds of lost time.

6 References

Project Description:

- [1] <http://www.cse.msu.edu/~cse435/Projects/F2016/ProjectDescriptions/Mobis-Ped-Avoid-2016-Agnew.pdf>

Project Website:

- [2] <http://www.cse.msu.edu/~cse435/Projects/F2016/Groups/PEDAC1/web/>

Class Website:

- [3] <http://cse.msu.edu/~cse435/>

Prototype Assets:

- [4] Car: <http://www.clipartkid.com/green-racing-car-top-view-by-qubodup-Y4dvlg-clipart/>

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.