Software Requirements Specification (SRS)

Backup Collision Avoidance System (BCAS)

Authors: Camille Lewis & Kira Chan
& Brenden Hein & Cameron White

Instructor: Dr. Betty H.C. Cheng
Michigan State University - CSE870

April 2021

1 Introduction

According to the Texas Department of Transportation, many fatal accidents in 2019 involved collisions with pedestrians when backing up [1]. While most drivers tend to be careful when operating a vehicle in reverse, there are more blind spots where pedestrians or objects may not be visible to the driver. Furthermore, children are often small and may not be visible behind the vehicle's rear [2]. As a result, drivers may not be aware of a child's presence behind the vehicle when backing up, causing an accident. Thus, new approaches are needed to ensure pedestrian safety to prevent backup collision accidents.

This document details the requirements, functionality, elements, and uses of the Backup Collision Avoidance System (BCAS) that assists automobile drivers from unknowingly backing into objects, animals, and/or people. In section 1, the purpose of the document and the scope of the software to be developed for BCAS are discussed.

1.1 Purpose

The purpose of the document is to provide easily understandable details of BCAS such that the document both satisfies the customer's requirements and is a clear guide for designing and implementing the system. The document is intended for the developers to thoroughly understand the BCAS before they begin designing and implementing the system. The document is also intended for the stakeholders of the project to understand all aspects of the system.

1.2 Scope

BCAS is an embedded system [3] in an automotive vehicle whose primary objective is to prevent collisions, injuries, and damages during the reverse operation of the vehicle.

As an embedded system, the BCAS to be developed obtains sensor values from onboard sensors, processes the data with the controller, and utilizes actuators to execute the necessary responses from the controller. System communications between the components are achieved using the Controller Area Network (CAN) bus [4].

The BCAS system uses sensors to detect obstacles in the path of the vehicle and responds with warnings to the driver and, if necessary, automatically stops the vehicle. A camera feed of the path behind the vehicle is displayed to the driver. The driver has options to override or disable the system. Secondary services of the system include illuminating the path behind the vehicle with a warning light and removing ice and dirt occluding the main backup camera with a heated compartment and a spray nozzle.

Several similar systems have been proposed or used in the past. Smithline [5] proposed a collision avoidance system that uses a variety of devices with range detecting capabilities to determine distance between a vehicle's rear and an object. Studt and Taylor [6] proposed a back-up aid system that uses similar range scanning sensors to determine the time to collision with nearby objects. The system then uses visual and audio signals to alert the driver of a potential collision if at least one of the sensors determines the time to collision is less than 5 seconds. These warning signals play at varying frequencies depending on the warning level. However, neither system employs machine learning techniques to enhance target detection nor provide automatic braking for the driver when a collision is imminent. Coelingh et al. [7] proposed an Automatic Emergency Braking (AEB) system intended to avoid or mitigate pedestrian accidents. Finally, Gandhi et al. [8] surveyed the research field for state-of-the-art techniques used to provide backup collision prevention.

1.3 Definition, acronyms, and abbreviations

- **Backup Collision Avoidance System (BCAS):** The BCAS is the entire embedded system described throughout the document.
- **Controller Area Network (CAN):** The CAN bus is a communication channel in automotive vehicles that enables communications between electronic control units, sensors, and actuators.
- **Dashboard:** The dashboard is the information display located behind the steering wheel where the speedometer and odometer are located.
- **Human Machine Interface (HMI):** The HMI is the infotainment display, generally mounted to the right of the steering wheel.
- **Obstacle:** An obstacle is anything the vehicle could collide with while backing up. Some examples includes poles, buildings, other vehicles, and pedestrians.

1.4 Organization

The remainder of the requirements document is organized as follows. First, Section 2 describes the BCAS system. Next, Section 3 enumerates the system requirements for the BCAS. Section 4 includes diagrams for the BCAS system and their descriptions that

model the operation of the system in detail. Finally, Section 5 demonstrates the proof of concept prototype for the BCAS.

2 Overall Description

Section 2 describes the BCAS in detail. First, the document summarizes the product perspective and product functions. Next, the expectations for the users of the system are described. The document then describes the constraints set on the system. Finally, the document explains the assumptions, dependencies, and approportioning of requirements.

2.1 Product Perspective

The BCAS is responsible for obstacle detection, collision prevention, and damage mitigation if an accident were to occur for objects residing behind a vehicle, as they may be out of view of the driver's line-of-sight. While backup systems can be an afterthought for many new vehicles, the potential damage that may be caused by accidentally backing into objects or pedestrians warrants a well-designed system, making the BCAS an important part of the vehicle. As accident prevention and damage mitigation are both high-level goals of vehicles, the BCAS works specifically to prevent unintended collisions with pedestrians or objects during the reverse operation of the vehicle. As such, the BCAS operates only when the vehicle is in reverse.

While the BCAS is a unique addition to a vehicle, it frequently interfaces with many other components of the vehicle. As a subsystem of the vehicle, the BCAS cannot exist independently as it relies on existing components on the vehicle. Some examples include the dashboard, the haptic feedback system, and the sound system, which are mainly used for warning the driver of an obstacle. The dashboard alerts the driver though visual cues with warning messages. The sound system warns the driver in the form of audible beeps of varying frequency, relative to the distance of the detected object or pedestrian. Finally, the haptic feedback system alerts the driver by vibrating the steering wheel. The Human Machine Interface (HMI) allows the driver to directly interact with the BCAS system and control certain aspects of it (such as cleaning the main camera or disabling the system). Finally, the BCAS uses the braking system to control the automatic braking in an attempt to prevent or mitigate damages from an accident. Warning or mitigation responses are decided by the BCAS with the BCAS controller. The controller obtains input from the system sensors and executes the corresponding response(s) or action(s).

Figure 1 shows the KAOS goal model [9] for the BCAS system. In the KAOS goal model, high level system objectives are decomposed into a hierarchical arrangement of lower level sub-goals or requirements. Sub-goal defines how the parents goal can be satisfied. *Goals* are represented by blue parallelogram, organized hierarchically. Hexagons denote *Agents* of the system (i.e., sensors of the system). Goal decompositions (also known as refinements) are denoted by arrows with a circle in between. If multiple refinements are connected from multiple sub-goals to a parent goal, then it is known as a *OR* refinement. On the other hand, *AND* refinement are represented by the merging of two sub-goal arrows into a single circle pointing to the parent goal. Yellow sub-text for a goal represent the utility function used to evaluate the satisficement (degree of

satisfaction from 0 to 1) of the goal. Figure 2 shows the legend for the KAOS goal model.

In the KAOS goal model for the BCAS, the high level goal to be satisfied is preventing backup collision. The goal is decomposed into multiple functional and non-functional sub-goals. First, the BCAS provides the camera feed to the driver. Next, the BCAS will not abruptly brake unless a collision is imminent. Finally, the BCAS will detect and mitigate imminent collisions by utilizing the different sensors in the vehicle (i.e., G3). Goal G3 is further decomposed to depict the different situations, where the object distance varies in multiple ranges (G5-G7). G8 represents the BCAS system detecting the distance of the nearby object. Further goal decompositions use sensor data to satisfy higher level goals.

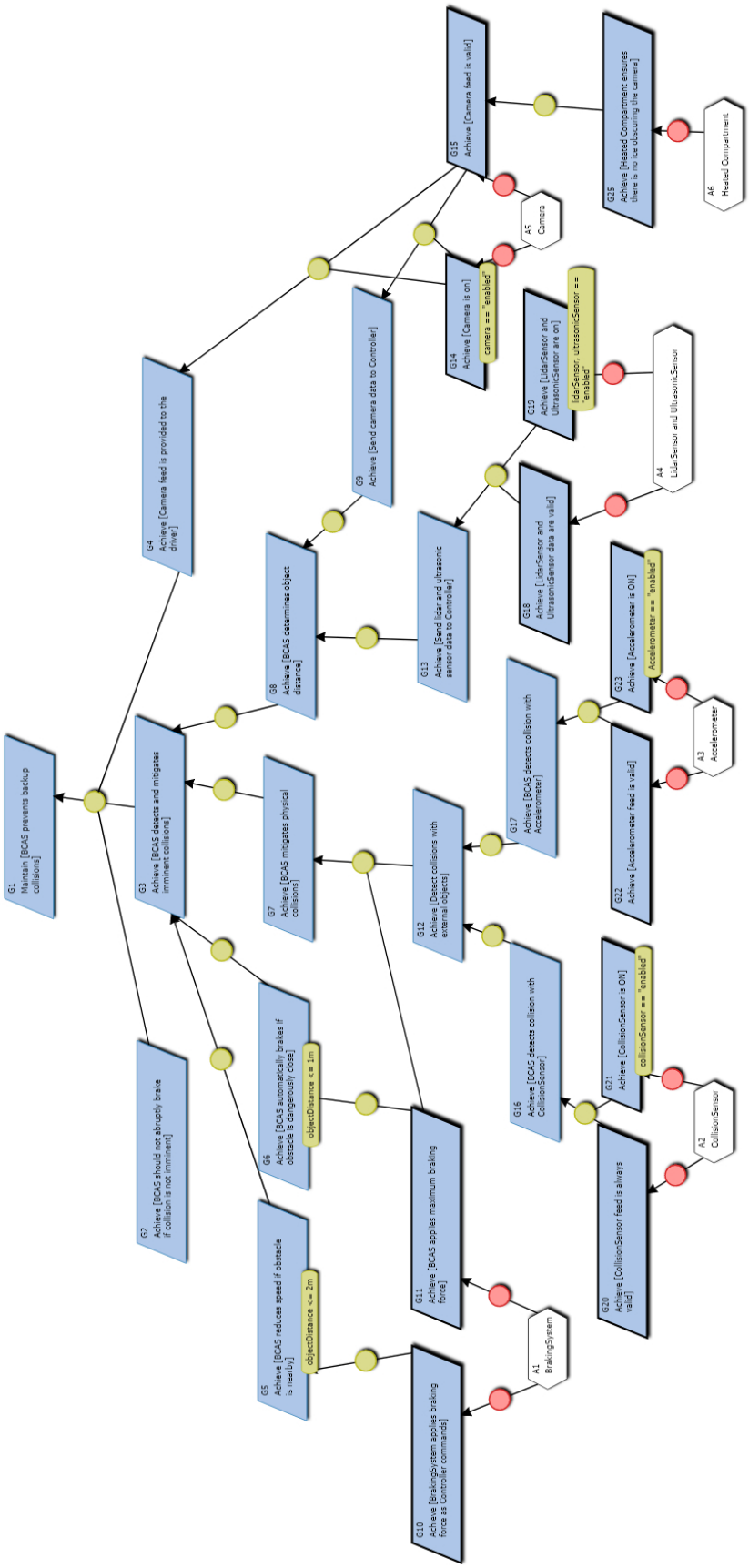

Figure 1: KAOS goal model for the BCAS

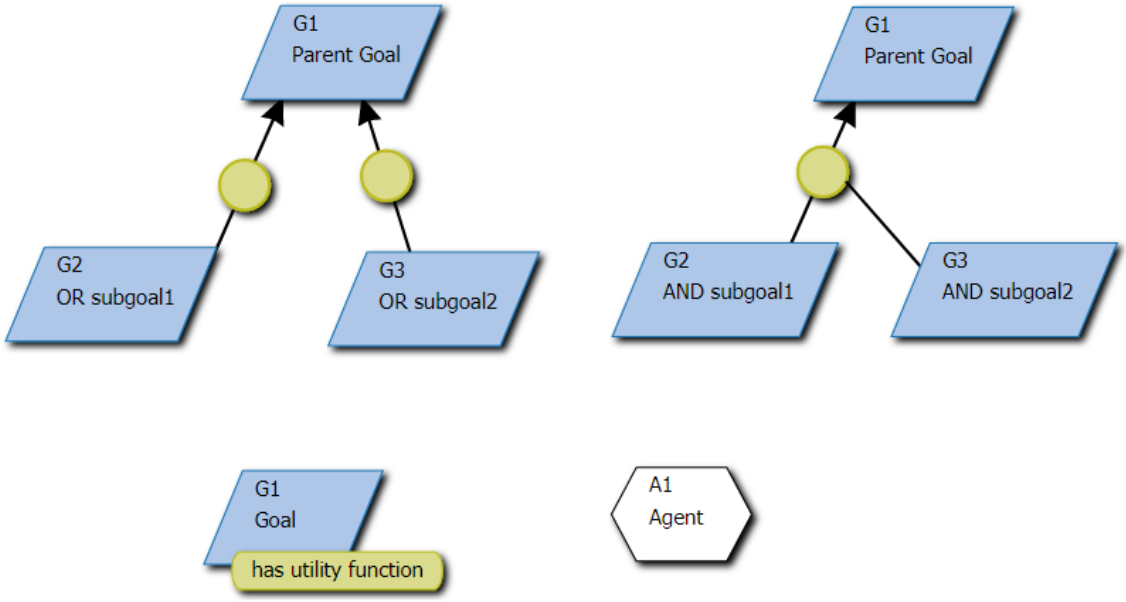

Figure 2: Legend key for the KAOS modeling notation

2.2 Product Functions

The BCAS provides an enhanced backup experience for the driver of the vehicle by increasing the driver's awareness of the vehicle's surroundings. The system's main objective is to prevent backup accidents during the operation of the vehicle. When the driver of the vehicle shifts the gear into reverse, the BCAS is automatically turned on. While the BCAS is active, the HMI uses the camera sensors located at the rear of the vehicle to display four separate camera feeds to the driver. The main backup camera is located above the license plate and displays a rear view of the vehicle's surroundings. The main camera contains a heated compartment that prevents ice from occluding the camera and a spray nozzle controlled by the HMI to remove debris on the camera. Two side view cameras display the camera feed of the vehicle's sides, while the 360-degree camera located at the roof of the vehicle's rear provides a "bird's-eye" point of view of the vehicle's surroundings. Using the camera displays from the HMI, the driver of the vehicle can ensure that there are no objects or pedestrians in the blind spots of the vehicle before backing up.

The BCAS system provides warning and mitigation for the driver. If an object or pedestrian is detected by the detection algorithm, then a thin box (red and green, respectively) highlights the object or pedestrian to notify the driver. While the BCAS is active, a pedestrian warning light is projected onto the road behind the vehicle to alert nearby pedestrians that the vehicle is backing up, as well as increasing visibility of objects or pedestrians to the driver. As the vehicle approaches an object, the BCAS provides a visual warning on the driver's dashboard, an audio warning, and haptic feedback on the steering wheel to alert the driver of a potential collision. The frequency and magnitude of the warnings increase as the vehicle approaches the object or pedestrian. If the vehicle approaches an object or pedestrian within 2 meters while backing up, then the BCAS shall reduce the current speed of the vehicle by 25% and display a message on the dashboard and HMI indicating that the driver should brake. The driver will also be notified that an object has been detected through an audio prompt in the form of beeps and by the vibration of the steering wheel. If the system detects that a collision is imminent (i.e., the vehicle is reversing within 1 meter of an object), then the BCAS shall apply the brakes to stop the vehicle and display a message to the user indicating that the system is automatically braking.

Finally, if the BCAS detects a collision using the contact sensors located on the rear bumper or by detecting unintended deceleration, then the BCAS mitigates the collision by applying the brakes with maximum force, which is intended to provide a fail-safe option for the BCAS should the vehicle fail to detect and prevent a collision with an object or pedestrian. The BCAS system can be overridden for the duration of the time the driver is in reverse by pressing a button on the steering wheel or completely disabled in the vehicle's settings.

2.3 User Characteristics

The expectations for a user of the BCAS system are described here. The user (driver) of the BCAS is expected to have passed the driving test and possess a valid driver's license. Before using the BCAS, the driver is expected to read the user manual provided that describes the features, uses, and operations of the system. No other background or skill level expectations are required for the operation of the system. During the operation of the vehicle and the BCAS system, the driver is ultimately responsible for following all traffic laws and ensuring the safety of the vehicle's passengers and nearby pedestrians.

2.4 Constraints

The constraints set for the BCAS are listed here. The system is active if and only if the vehicle is in reverse gear. While the system is active, the visibility of the object or pedestrian from the camera feed must be significant enough to allow for the detection algorithm to identify the object or pedestrian. The object or pedestrian attempting to be detected by the the vehicle's ultrasonic sensors must not contain sound-absorbing material. The vehicle's lidar sensor will only be effective during clear weather conditions (i.e., not rain, snow, fog, etc.). Similarly, the sensors of the system must not be obstructed by another object. The driver of the vehicle is expected to clear the camera of obstruction if the view is obstructed. The system may still detect objects that are not a threat to the vehicle, such as potholes or objects being hitched to the back of the vehicle, and the system should be able to be overridden in such case. The BCAS may not detect objects or pedestrians 100% of the time, so if a collision occurs, the system will attempt to mitigate damages by applying the maximum braking force. The BCAS must be able to communicate with the braking system in case a potential collision is detected to allow the BCAS to automatically brake. The BCAS should be able to connect with the HMI so the driver can interact with the system. The vehicle must have a haptic feedback system built into the steering wheel. When the BCAS detects a potential collision, the driver must be notified so they do not work against the system. In the case of a component failure for the BCAS system, the driver will be notified of the issue.

2.5 Assumptions and Dependencies

The BCAS to be developed uses other systems and components of the vehicle. Assumptions and dependencies are made for the hardware components, software components, and user behavior. First, external hardware or software systems of the vehicle, such as the braking system, HMI, sound system, etc. are expected to be installed on the vehicle and are functional. When the BCAS requests a service provided by other systems (e.g., sending haptic feedback to the user), the corresponding systems are expected to perform the action requested by the BCAS. Furthermore, since the BCAS uses multiple sensors and feedback systems, all external communication formats are assumed to be uniform (e.g., all sensor readings are returned in the same format). Finally, the driver of

the vehicle is expected to use the BCAS system as specified throughout the document.

2.6 Apportioning of Requirements

Several components of the BCAS system will be considered to be beyond the scope of the project and may be addressed in the future. First, the pedestrian or object detection algorithm used for object recognition is not considered in the document. The algorithm is assumed to be a black-box (i.e., the algorithm's implementation is abstracted or hidden). The data stream provided by the sensors will not be fully detailed in the project, as well. Finally, the vehicles underlying operating system and much of the resource management (memory, storage, networks, etc.) will not be included in the project.

3 Specific Requirements

Section 3 describes the hardware, invariants, and system requirements for the BCAS to be developed.

3.1 Hardware Requirements

The following section overviews hardware components that are required for the system. Hardware installed on the vehicle should originate from or be approved by the Original Equipment Manufacturer (OEM) to ensure compatibility. As the BCAS's hardware requirements may intersect (overlap) with hardware that already exist and used in other systems on the vehicle, intersecting hardware can be reused with adaptations to the BCAS.

1. Hardware already present on vehicle
    - Brakes
    - Acceleration and speed sensor
    - Dashboard (refers to the display behind the steering wheel, where the speedometer and odometer are displayed)
    - Human Machine Interface (HMI) (refers to the infotainment display, often located to the right of the steering wheel)
    - Haptic feedback device built into the steering wheel
    - Communication channel (Controller Area Network (CAN) bus)
2. Hardware specific for BCAS
    - Main backup camera located above the license plate
    - Side-view cameras on left and right side of the vehicle's rear
    - 360-degree wide-lens camera located at the highest point of the vehicle's rear
    - Ultrasonic sensor (used to detect objects moving towards the vehicle)
    - Lidar sensor (used to detect objects moving towards the vehicle)
    - Sound alarm system
    - Collision sensor located on rear bumper of the vehicle
    - Override button located on the steering wheel with a red LED

3.2 Invariants

The invariants section provides an enumeration of the invariant requirements for the BCAS. Invariants are a special subset of requirements that must always be true. If an invariant is violated, then the system has failed.

A. Prevent injuries.
B. The vehicle cannot exceed a maximum speed of 5 meters/second while the system is active.
C. The system must correctly detect obstacles.
D. The vehicle always stops within 1 meter of a stationary object, unless overridden by the driver and/or the detected object is already less than 1 meter away.
E. The driver can override the system at any time by pressing a button designated for the BCAS on the steering wheel. The system resumes when the same button is pressed again, or if the car returns to reverse gear from another gear position. The button is illuminated with a red LED to indicate that the system is temporarily suspended.
F. If the system detects an obstacle, then the system will alert the driver.
G. The failure of a single sensor will not hinder sensor feedback from other non-damaged sensors to the driver.

3.3 System Requirements

The following section enumerates the system requirements for the BCAS to be developed.

1. The system *turns on* automatically when the driver puts the vehicle in the reverse (R) gear.
   (a) (Non-functional) The system should boot up and be ready for operation in less than 2 seconds.
2. When the system is active, the HMI displays the camera feeds to the driver.
   (a) The main backup camera, located above the license plate, is displayed on the top half of the HMI. The main backup camera is denoted by **M**.
   (b) The cameras located on the sides of the vehicle capture input from the left and right side of the vehicle's environment, denoted by **L** and **R** respectively. The feed for the left and right cameras are displayed on either side of the main backup camera feed.
   (c) A 360-degree wide-lens camera, located at the highest point of the vehicle's rear, captures the environment directly behind the vehicle with a "birds eye view". The feed is displayed on the bottom half of the HMI, denoted by **W**.
   (d) The matrix below shows an example of how the camera feeds are displayed on the HMI to the driver.

$$\begin{bmatrix} \mathbf{L} & \mathbf{M} & \mathbf{R} \\ \Leftarrow & \mathbf{W} & \Rightarrow \end{bmatrix}$$

   (e) The main backup camera is installed inside of a heated compartment that prevents the camera from being occluded by ice. The compartment turns

on and begins monitoring the surrounding temperature when the vehicle is turned on. The heating element turns on when the temperature is 35°F or below.

(f) The main backup camera is installed behind a water-repellent glass, tilted at a 60 degree angle, as shown in Figure 3.

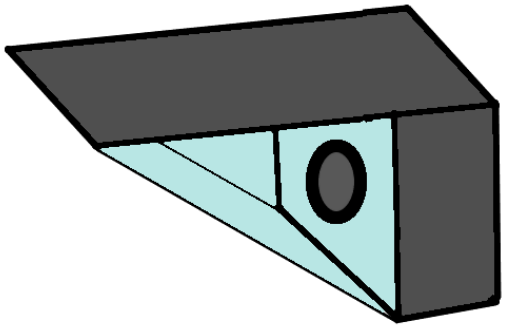

Figure 3: Example of the main backup camera used in the BCAS

(g) A spray nozzle is located by the camera that can be activated by the driver by pressing a button on the touch screen of the HMI to spray dust or debris off the camera lens.

(h) (Non-functional) The main backup camera must have a resolution of at least one megapixel.

(i) (Non-functional) The camera refresh rate must be higher than 60 frames per second.

(j) (Non-functional) The HMI must have a minimum resolution of 720p.

(k) (Non-functional) The HMI must be at least nine centimeters in height [10].

3. When the system is active, a pedestrian warning light is projected onto the road behind the vehicle to alert nearby pedestrians.
4. The system detects potential objects or pedestrians.
   (a) The system uses input from the cameras to detect objects or pedestrians by passing their input feed through a pattern recognition or machine learning algorithm.
      i. The system highlights detected pedestrians in a thin green box.
      ii. The system highlights detected objects in a thin red box.
   (b) Using the ultrasonic sensors, infrared sensors, and the lidar sensor, the system detects objects or pedestrians moving towards the vehicle's rear.
      i. Four ultrasonic sensors are installed at the rear of the vehicle. Two of these sensors are installed on both sides of the license plate and two sensors are installed at the bottom corners of the vehicle's rear.
      ii. The system uses arrows on the camera feeds to indicate moving objects' direction.
5. If the vehicle approaches an object or pedestrian while backing up, then the vehicle

provides the driver with audio and haptic feedback.
   (a) If the vehicle is within 5 meters of an object or pedestrian, then the system beeps once every 1/4 of a second.
   (b) When the vehicle gets within 2 meters of an object or pedestrian, the system scales up the beeping speed to once every 1/10 of a second.
   (c) If the vehicle is within 2 meters of an object or pedestrian, then the system will provide haptic feedback to the driver by vibrating the steering wheel. The system will also display a "AUTOMATICALLY BRAKING" message on the dashboard and the HMI.
6. When the vehicle is likely to collide with an object or pedestrian, calculated by the current vehicle speed and object's distance, the system automatically applies the brakes to slow down the vehicle or stop the vehicle completely according to the following conditions. The driver of the vehicle may override the automatic braking after the vehicle has completely stopped by applying the gas pedal after three seconds.
   (a) If the vehicle is within 2 meters of an object or pedestrian, then it reduces the current speed of the vehicle by 25%.
   (b) If the vehicle is within 1 meter of an object or pedestrian, then it applies the maximum braking force to stop the vehicle.
   (c) When the vehicle is automatically braking, a message of "AUTOMATICALLY BRAKING" should indicate so on the dashboard and the HMI.
   (d) The driver can override the automatic braking by applying pressure to the gas pedal after the vehicle has been stationary for 3 seconds.
7. When the vehicle detects a collision with external environment objects or pedestrians, the system stops the vehicle with maximum braking force within 0.1 seconds, providing a failsafe option for the BCAS should the vehicle fail to detect an object or pedestrian with the camera, ultrasonic, and lidar sensors.
   (a) The system detects a collision with external variables using the contact sensors installed on the rear bumper of the vehicle. The system applies maximum braking force and stops the vehicle when the contact sensors have been triggered.
   (b) The system detects a collision using the speed/acceleration sensor. When sudden deceleration is detected without driver input (i.e., the vehicle has likely hit an external variable), the system applies maximum braking force and stops the vehicle.
8. If the sensor(s) of the vehicle is/are occluded, compromised, or fails, then a message is displayed to the driver via the Dashboard and the HMI to remove the occlusion or alert the driver in case of an attack or failure.
   (a) While the sensors cannot function correctly, the system is temporarily suspended. Non-damaged components of the system that do not impact the operation of the vehicle, such as camera sensor input, will continue to provide feedback to the driver.
   (b) If a sensor is permanently damaged, then a message asks the driver to bring

their vehicle to a certified dealer for repair.

9. The BCAS can be manually disabled or overridden, which is present for the driver if the system has been compromised.
   (a) The driver of the vehicle can turn off the system in the vehicle's *Settings* menu via the HMI while the vehicle is in park (P). The system will continue to provide the driver with camera, ultrasonic, and lidar input, but will not automatically brake for the driver. Sound and haptic feedback will also be disabled.
   (b) During the parking operation of the vehicle (i.e., the car is in reverse (R)), the driver can temporarily override the system by pressing a button present on the driver's steering wheel designated for the BCAS. The system resumes the next time the vehicle is put in reverse from another gear position, or if the driver presses the button again. The button is illuminated by a red LED while the BCAS is suspended.
10. (Non-functional) The system should not be easily compromised by adversaries.
   (a) The number of outward facing sensors should be minimized while BCAS is active.
   (b) Sensors that do not require external communication, such as the cameras or lidar sensor, should not have wireless communication capabilities.
   (c) Any non-essential outward facing communication devices that are not used by the BCAS (e.g., the GPS sensor) should be temporarily disabled while the system is active to prevent cyber attacks.
   (d) Failsafe - when components of the BCAS fail, the system should have fail-safe capabilities to ensure driver safety and continue to provide sensor input feedback.
11. (Non-functional) BCAS should not cause major distraction or annoyance to the driver.
   (a) Audio alarm should not exceed 75 dBA.
   (b) The radio and other audio system should be reduced to 25% of their maximum volume when the BCAS is active. The volume returns to the original value when the system turns off.
   (c) Automatic braking of the vehicle while the object is more than 1 meter away should not be abrupt.
12. Hardware and software testing requirements will be added in a future iteration.

4 Modeling Requirements

Section 4 presents various diagrams and their corresponding descriptions to depict key elements, interactions, scenarios, and services of the BCAS system in detail.

4.1 Use Case Diagram and Descriptions

The following sections introduce the use case diagram for the BCAS and the corresponding use case descriptions. A use case diagram captures a user's visible function and interaction with the system. The use case descriptions describe each use case in the use case diagram in detail.

4.1.1 Use Case Diagram

Figure 4 shows the use case diagram for the BCAS system to be developed, where stick figures represent actors, the system boundary is represented by a rectangle, circles inside a system boundary denote use cases, and line connections represent associations between actors and use cases. Arrows with *Include* or *Extend* tag represent special relationships between the two use cases. In an include relationship, multiple classes use the functionality of the included use case, leading to an includes relationship where the included use case can be reused. An extended use case represents a use case similar to another use case, but extends the original use case with additional features or functionalities. In the use case diagram for the BCAS, the driver is the primary actor, while **the BCAS forms the system boundary**.

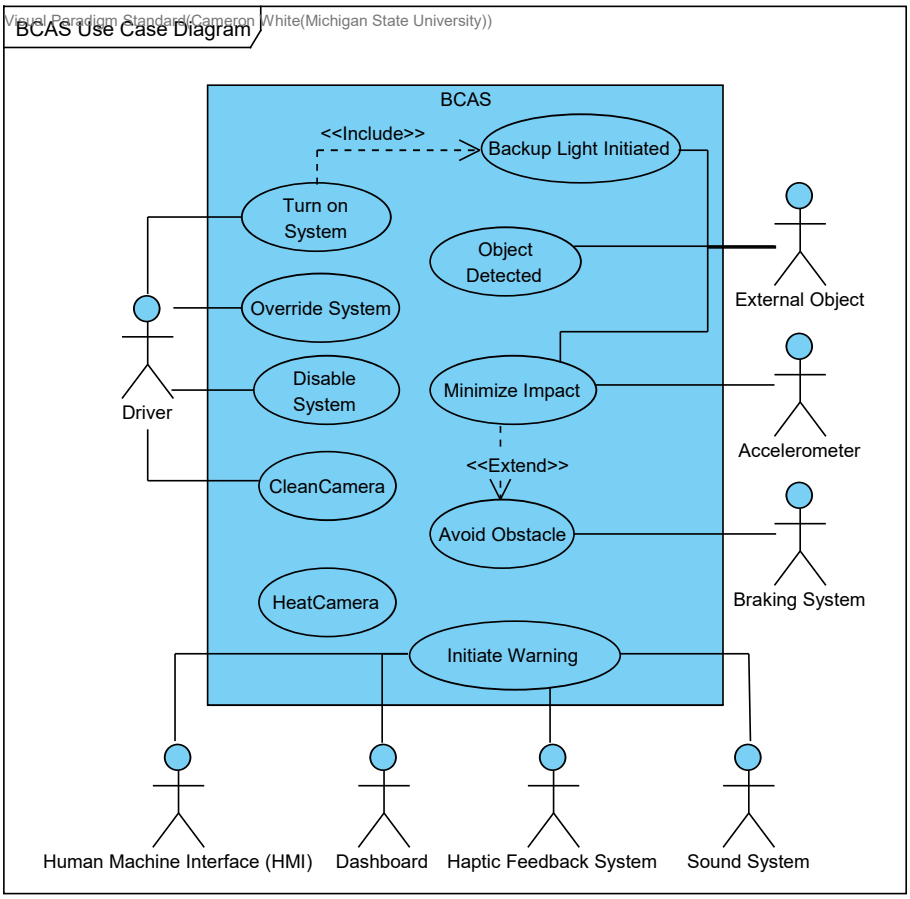

Figure 4: Use case diagram for BCAS

4.1.2 Use Case Descriptions

Tables 1 through 10 detail the use cases in Figure 4, providing a comprehensive description for the use cases. The use case field includes the name of the use case. The actor field denotes the actors involved in the use case. The description provides a detailed overview for the use case. The type categorizes the use case into primary or secondary. Includes and extends list other use cases that are related to the use case. Cross-ref enumerates the invariants and requirements in Section 3 that are referenced in the use case. Finally, the use cases field describes other use cases that must be satisfied for the current use case to be active.

Use Case:	Turn on System
Actors:	Driver
Description:	Once the vehicle is put into reverse gear, the system is turned on. While the system is turning on, each of the system components is checked for a hardware failure. If a hardware failure is detected, then the system is put into a suspended state. While the system is suspended, automatic braking is disabled, but all undamaged sensors still provide feedback to the driver.
Type:	Primary
Includes:	Backup Light Initiated
Extends:	N/A
Cross-refs:	Requirements 1a, 2a, 2b, 2c, 2d, 2e, 2f, 2g, 2h, 2i, 2j, 2k, 8a, 10a, 10b, 10c, 10d, 11b, Invariants A, B, G
Use cases:	N/A

Table 1: Use case description for *Turn on system*

Use Case:	Override System
Actors:	Driver
Description:	When the system is active, the driver can override the system at any time by pressing the designated override button. When the override button is pressed, the system is temporarily suspended and can be re-engaged by pressing the button again. The button is illuminated by a red LED while the system is suspended. Furthermore, the system is no longer suspended the next time the vehicle shifts into reverse gear. When the system automatically brakes, the driver may apply pressure to the gas pedal to regain control of the vehicle after the vehicle has been stationary for 3 seconds.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirements 6d, 9b, Invariant E
Use cases:	The system must be active

Table 2: Use case description for *Override System*

Use Case:	Disable System
Actors:	Driver
Description:	The driver of the vehicle can turn off the system in the vehicle's Settings menu via the HMI while the vehicle is in park (P). The system will continue to provide the driver with camera, ultrasonic, or lidar input, but will not automatically brake for the driver. Sound and haptic feedback will also be disabled. The system will remain disabled until the driver re-enables it in the Settings menu.
Type:	Primary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 9a
Use cases:	N/A

Table 3: Use case description for *Disable System*

Use Case:	Initiate Warning
Actors:	HMI, Dashboard, Haptic Feedback System, Sound System
Description:	If the vehicle is within 5 meters of an object, then the system beeps once every 1/4 of a second. As the vehicle's distance to the object decreases to 2 meters or less, the system beeps at an increasing frequency of 1/10 of a second per beep. If the vehicle is within 2 meters of an object, then the system will provide haptic feedback on the steering wheel. When the system is automatically braking, a message of "AUTOMATICALLY BRAKING" will display on the dashboard and HMI. If a hardware component of the system has failed, then a warning on the HMI and dashboard will display indicating the failure and instructing the driver to bring their vehicle in for repair. The audio warning should not exceed 75 dBA.
Type:	Primary & Essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirements 5a, 5b, 5c, 6c, 8b, 11a, Invariant F
Use cases:	N/A

Table 4: Use case description for *Initiate Warning*

Use Case:	Minimize Impact After Collision Detected
Actors:	External Object, Accelerometer
Description:	The system detects a collision with external objects with the contact sensors installed on the rear bumper of the vehicle. The system also detects collisions by detecting an unintended deceleration through the acceleration sensor. If the system detects a collision, then the BCAS applies maximum braking force within 0.1 seconds to minimize further damage.
Type:	Primary & Essential
Includes:	N/A
Extends:	Avoid Obstacle
Cross-refs:	Requirements 7a, 7b
Use cases:	N/A

Table 5: Use case description for *Minimize Impact After Collision Detected*

Use Case:	Object Detected
Actors:	External Object
Description:	Using a pattern recognition or a machine learning algorithm, the system uses the input feeds from the cameras to detect external objects and highlights pedestrians in green and stationary objects in red. Using ultrasonic and lidar sensors, the system detects objects or pedestrians moving toward the vehicle's rear and uses arrows on the camera feeds to indicate the moving objects' direction.
Type:	Primary & Essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirements 4a, 4b, 8a, 10d Invariants C, G
Use cases:	N/A

Table 6: Use case description for *Object Detected*

Use Case:	Avoid Obstacle
Actors:	Braking System
Description:	If the vehicle is within 2 meters of an object or a pedestrian, then the system reduces the speed of the vehicle by 25%. If the vehicle is within 1 meter of an object or pedestrian, then the system applies maximum braking force to stop the vehicle.
Type:	Primary & Essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirements 6a, 6b, 11c, Invariants A, B, D
Use cases:	The system must detect a collision, an object, or unintended deceleration to automatically reduce the vehicle speed (i.e., *Object Detected* or *Minimize Impact After Collision Detected* active).

Table 7: Use case description for *Reduce Vehicle Speed*

Use Case:	Backup Light Initiated
Actors:	External Object
Description:	When the system is active, a backup indication light is projected onto the road behind the vehicle to alert nearby pedestrians.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 3
Use cases:	N/A

Table 8: Use case description for *Backup Light Initiated*

Use Case:	Clean Camera
Actors:	Driver
Description:	When the system is active, the driver has the ability to press the clean camera button which will clean the main camera with a continuous spray of cleaning fluid until the driver releases the button.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 2g
Use cases:	N/A

Table 9: Use case description for *Clean Camera*

Use Case:	Heat Camera
Actors:	N/A
Description:	When the vehicle is turned on, the heated compartment that contains the main camera is turned on and continually heats the camera when needed.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 2e
Use cases:	N/A

Table 10: Use case description for *Heat Camera*

4.2 Domain Model and Data Dictionary

The domain model for the BCAS and the corresponding data dictionary are introduced here. The domain model uses class diagram notation to depict the key elements of the system and the interactions between those elements. The data dictionary provides an in-depth description of each class, its attributes, its operations, and the relationships between those classes.

4.2.1 Domain Model

The domain model in Figure 5 shows the elements of the BCAS system and how they interact with each other. The two blue boxes are separate packages: one for the elements of the BCAS embedded system and one for vehicle components outside the system and the driver. Each box in the diagram is a class which is an individual object in the system. Each class may have its own attributes, which are saved values, and operations, which are means of interacting with other classes. A line connecting two classes describes a relationship between the two classes and can be one of the following types. A line with no extraneous symbols is an association, which defines a conceptual connection between two classes. A label and directional arrow on an association further describes the relationship between the two classes and can be read as a sentence following the direction of the arrow. For example, in Figure 5 the Controller turns on/off the PedestrianWarningLight. A line with a diamond on one end is a composition relationship which describes that a class can be made up of instances of other classes. An illustration of the composition relationship is shown in Figure 5 where the MainCamera class is composed of the HeatedCompartment class and the SprayNozzle class. A line with a triangle on one end describes a generalization or inheritance relationship in which a child class is a more specialized instance of a parent class. For instance, in Figure 5 MainCamera, SideCamera, and 360Degree Camera are derived from the CameraSensor class.

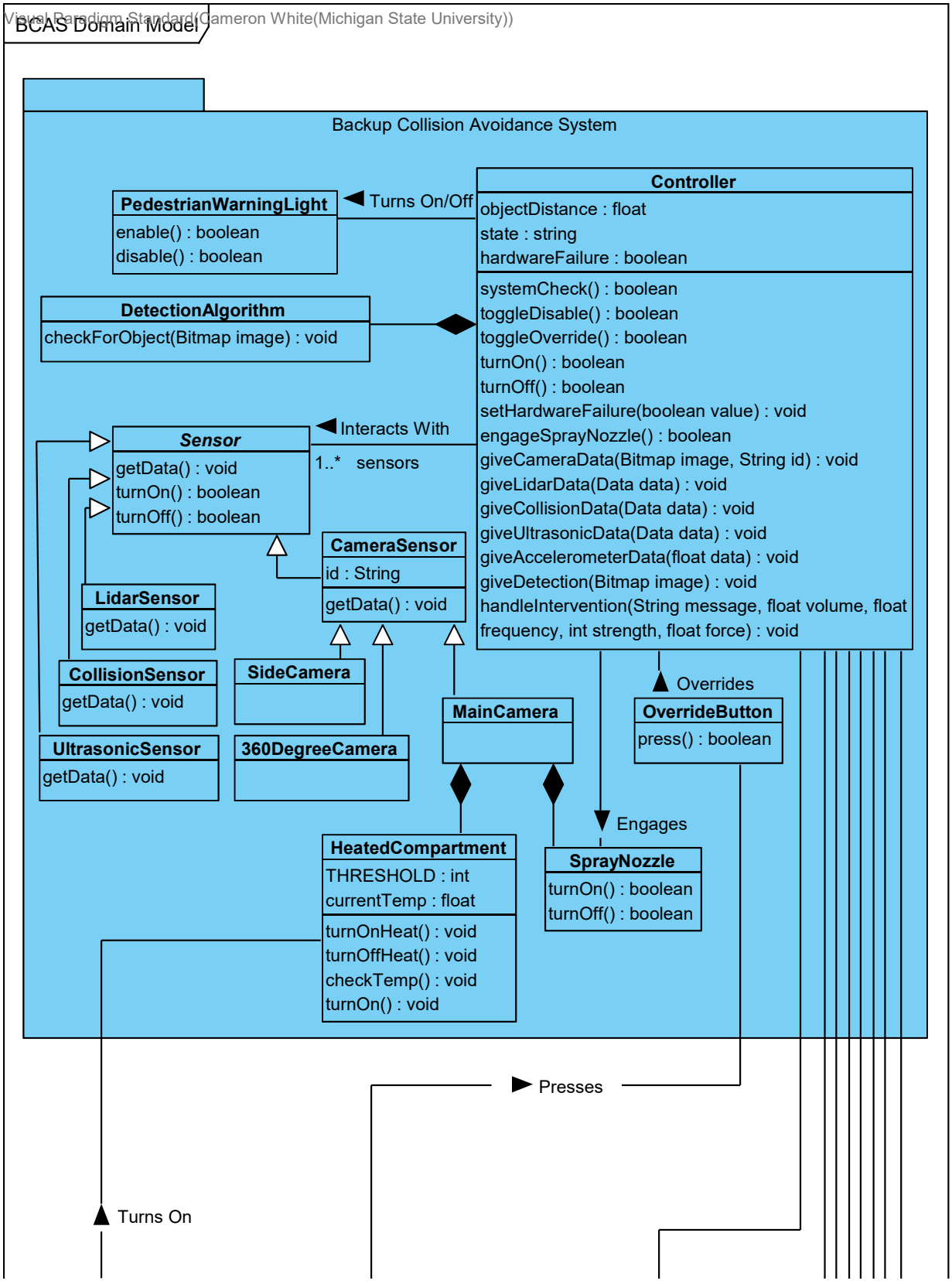

Figure 5: High-Level Class diagram (Domain Model) for BCAS

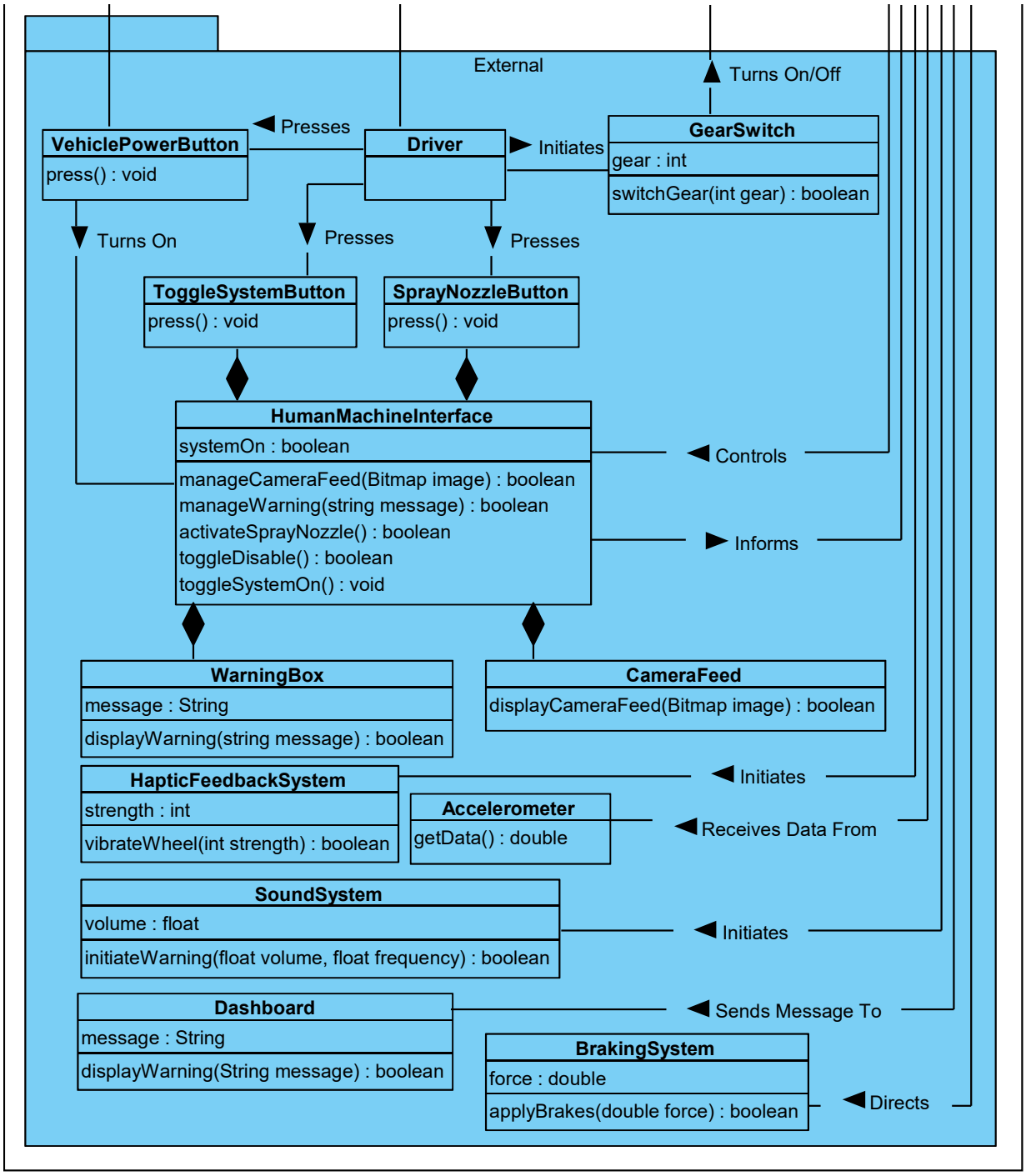

Figure 5: High-Level Class diagram (Domain Model) for BCAS (cont.)

4.2.2 Data Dictionary

Tables 11 through 37 further describe each of the classes present in Figure 5 by detailing the relationships, the attributes, the operations, and the purpose of the class. There are three types of relationships a class can have with another class which are aggregation, association, and generalization. Association describes a mutual interrelation between two classes where both classes benefit from the connection. An aggregation represents owning another class or the owned class being a component of the owning class. Generalization depicts a general class that other classes then inherit from and these classes are known as parent and child classes respectively.

<table>
<tr><th>Class</th><td colspan="2"></td></tr>
<tr><td rowspan="5">Controller</td><td colspan="2">Description: The controller receives data from sensors, makes decisions, and calls functions of other system elements to enact response actions. The controller keeps track of the most recent nearest object distance measurement received from the sensors, the state of the system (enabled, suspended, or disabled), and whether or not there has been a hardware failure.</td></tr>
<tr><td colspan="2">Export Control: Private</td></tr>
<tr><td rowspan="3">Relationships</td><td>Associations: PedestrianWarningLight, Sensor, SprayNozzle, OverrideButton, GearSwitch, HumanMachineInterface, HapticFeedbackSystem, Accelerometer, SoundSystem, Dashboard, and BrakingSystem</td></tr>
<tr><td>Aggregations: Has DetectionAlgorithm as a part</td></tr>
<tr><td>Generalization: None</td></tr>
<tr><td colspan="2">Attributes:
objectDistance : float
state : string
hardwareFailure : boolean</td></tr>
<tr><td colspan="2">Operations:
systemCheck() : boolean
toggleDisable() : boolean
toggleOverride() : boolean
turnOn() : boolean
turnOff() : boolean
setHardwareFailure(boolean value) : void
engageSprayNozzle() : boolean
giveCameraData(Bitmap image, String id) : void
giveLidarData(Data data) : void
giveCollisionData(Data data) : void
giveUltrasonicData(Data data) : void
giveAccelerometerData(float data) : void
giveDetection(Data data) : void
handleIntervention(String message, float volume, float frequency, int strength, float force) : void</td></tr>
</table>

Table 11: Data dictionary entry for *Controller*

<table>
<tr><th>Class</th><td colspan="2"></td></tr>
<tr><td rowspan="6">PedestrianWarning Light</td><td colspan="2">Description: The PedestrianWarningLight is responsible for turning on a light that illuminates the path behind the vehicle.</td></tr>
<tr><td colspan="2">Export Control: Private</td></tr>
<tr><td rowspan="3">Relationships</td><td>Associations: Controller</td></tr>
<tr><td>Aggregations: None</td></tr>
<tr><td>Generalization: None</td></tr>
<tr><td colspan="2">Attributes: None</td></tr>
<tr><td colspan="2">Operations:
• enable() : boolean
• disable() : boolean</td></tr>
</table>

Table 12: Data dictionary entry for *PedestrianWarningLight*

<table>
<tr><th>Class</th><td colspan="2"></td></tr>
<tr><td rowspan="6">DetectionAlgorithm</td><td colspan="2">Description: The DetectionAlgorithm is responsible for carrying out the machine learning algorithm given data from a camera to detect pedestrian or obstacles.</td></tr>
<tr><td colspan="2">Export Control: Private</td></tr>
<tr><td rowspan="3">Relationships</td><td>Associations: None</td></tr>
<tr><td>Aggregations: Is part of Controller</td></tr>
<tr><td>Generalization: None</td></tr>
<tr><td colspan="2">Attributes: None</td></tr>
<tr><td colspan="2">Operations: checkForObject(Bitmap Image) : Bitmap</td></tr>
</table>

Table 13: Data dictionary entry for *DetectionAlgorithm*

Class		
Sensor	**Description:** The Sensor class is the base class for all sensors in the BCAS that are responsible for detecting objects located near the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: Parent class for UltrasonicSensor, CollisionSensor, LidarSensor, and CameraSensor
	Attributes: None	
	Operations: • getData() : void • turnOn() : boolean • turnOff() : boolean	

Table 14: Data dictionary entry for *Sensor*

Class		
UltrasonicSensor	**Description:** The UltrasonicSensor class inherits from the Sensor class and representing the Ultrasonic Sensors located at the rear of the vehicle that are used to detect objects using ultrasonic waves.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from Sensor
	Attributes: None	
	Operations: getData() : void	

Table 15: Data dictionary entry for *UltrasonicSensor*

Class		
CollisionSensor	**Description:** The CollisionSensor class inheriting from the Sensor class and represents the collision sensors located at the rear of the vehicle. The CollisionSensor is a pressure sensor and detects if the vehicle collides with an external object.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from Sensor
	Attributes: None	
	Operations: getData() : void	

Table 16: Data dictionary entry for *CollisionSensor*

Class		
LidarSensor	**Description:** The LidarSensor class inherits from the Sensor class and represents the lidar sensor located at the rear of the vehicle, using lidar (Light Detection and Ranging) technology to detect objects located behind the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from Sensor
	Attributes: None	
	Operations: getData() : void	

Table 17: Data dictionary entry for *LidarSensor*

Class		
CameraSensor	**Description:** The CameraSensor class inheriting from the Sensor class represents the camera sensors located at the rear of the vehicle. The camera collects a camera feed from a specific location of the vehicle to help detect objects near the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from Sensor. Parent class for MainCamera, SideCamera, and 360DegreeCamera
	Attributes: id : string	
	Operations: getData() : void	

Table 18: Data dictionary entry for *CameraSensor*

Class		
360DegreeCamera	**Description:** The 360DegreeCamera class inheriting from the CameraSensor class represents the 360 degree camera located on top of the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from CameraSensor.
	Attributes: None	
	Operations: None	

Table 19: Data dictionary entry for *360DegreeCamera*

Class		
SideCamera	**Description:** The SideCamera class inheriting from the CameraSensor class represents the cameras located on the sides of the vehicle that are used to see objects that are not directly behind the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: None
		Generalization: Inherits from CameraSensor.
	Attributes: None	
	Operations: None	

Table 20: Data dictionary entry for *SideCamera*

Class		
MainCamera	**Description:** The MainCamera class inheriting from the CameraSensor class represents the camera located on the rear of the vehicle that is used to see objects directly behind the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: Has parts HeatedCompartment and SprayNozzle
		Generalization: Inherits from CameraSensor.
	Attributes: None	
	Operations: None	

Table 21: Data dictionary entry for *MainCamera*

<table>
<tr><th>Class</th><td colspan="2"></td></tr>
<tr><td rowspan="5">HeatedCompartment</td><td colspan="2">Description: The HeatedCompartment maintains temperature for the camera's glass compartment. When the temperature is below a certain threshold, the HeatedCompartment system turns on and heats the glass camera compartment.</td></tr>
<tr><td colspan="2">Export Control: Private</td></tr>
<tr><td rowspan="3">Relationships</td><td>Associations: None</td></tr>
<tr><td>Aggregations: Is part of MainCamera</td></tr>
<tr><td>Generalization: None</td></tr>
<tr><td colspan="2">Attributes:
THRESHOLD : int
currentTemp : float</td></tr>
<tr><td colspan="2">Operations:
turnOnHeat() : void
turnOffHeat() : void
checkTemp() : void
turnOn() : void</td></tr>
</table>

Table 22: Data dictionary entry for *HeatedCompartment*

<table>
<tr><th>Class</th><td colspan="2"></td></tr>
<tr><td rowspan="5">SprayNozzle</td><td colspan="2">Description: The SprayNozzle handles cleaning the camera's glass compartment by spraying the glass with windshield wiper fluid.</td></tr>
<tr><td colspan="2">Export Control: Private</td></tr>
<tr><td rowspan="3">Relationships</td><td>Associations: Controller</td></tr>
<tr><td>Aggregations: Is part of MainCamera</td></tr>
<tr><td>Generalization: None</td></tr>
<tr><td colspan="2">Attributes: None</td></tr>
<tr><td colspan="2">Operations:
turnOn() : boolean
turnOff() : boolean</td></tr>
</table>

Table 23: Data dictionary entry for *SprayNozzle*

Class		
OverrideButton	**Description:** The OverrideButton allows the driver to suspend the backup system while backing up. The OverrideButton resumes the BCAS if the button is pressed again, or if the driver shifts gears into reverse again.	
	Export Control: Private	
	Relationships	**Associations:** Controller and Driver
		Aggregations: None
		Generalization: None
	Attributes: None	
	Operations: press() : boolean	

Table 24: Data dictionary entry for *OverrideButton*

Class		
VehiclePowerButton	**Description:** The VehiclePowerButton is the mechanism by which the driver turns on the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** HeatedCompartment, Driver, and HumanMachineInterface
		Aggregations: None
		Generalization: None
	Attributes: None	
	Operations: press() : void	

Table 25: Data dictionary entry for *VehiclePowerButton*

Class		
Driver	**Description:** The Driver is an external entity controlling the vehicle. They are responsible for the parking operation of the vehicle.	
	Export Control: Private	
	Relationships	**Associations:** VehiclePowerButton, OverrideButton, GearSwitch, ToggleSystemButton, and SprayNozzleButton
		Aggregations: None
		Generalization: None
	Attributes: None	
	Operations: None	

Table 26: Data dictionary entry for *Driver*

Class		
GearSwitch	**Description:** The GearSwitch class turns the system on when the driver puts the vehicle in reverse.	
	Export Control: Private	
	Relationships	**Associations:** Driver and Controller
		Aggregations: None
		Generalization: None
	Attributes: gear : int	
	Operations: switchGear(int gear) : boolean	

Table 27: Data dictionary entry for *GearSwitch*

Class		
HumanMachine Interface	**Description:** The HumanMachineInterface displays the camera feed to the driver when the BCAS system is active. Furthermore, warning messages such as "Automatic Braking" notifies the driver of potential danger of mitigation actions from the system. If a sensor is occluded, then the HMI also displays a message to prompt the user to clear the occlusion. Finally, a button on the HMI activates the spray nozzle to spray debris off the main camera lens.	
	Export Control: Private	
	Relationships	**Associations:** Driver and Controller
		Aggregations: None
		Generalization: None
	Attributes: systemOn : boolean	
	Operations: • manageCameraFeed(Bitmap image) : boolean • manageWarning(string message) : boolean • activateSprayNozzle(boolean toggle) : boolean • toggleDisable() : boolean • toggleSystemOn() : void	

Table 28: Data dictionary entry for *HumanMachineInterface*

Class		
HapticFeedback System	**Description:** The HapticFeedbackSystem is responsible for vibrating the steering wheel when the vehicle is within 2 meters of an obstacle.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: None
	Attributes: strength : int	
	Operations: vibrateWheel(int strength) : boolean	

Table 29: Data dictionary entry for *HapticFeedbackSystem*

Class		
Accelerometer	**Description:** The Accelerometer captures the current acceleration (or deceleration) rate of the vehicle and returns the value to the controller.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: None
	Attributes: None	
	Operations: getData() : double	

Table 30: Data dictionary entry for *Accelerometer*

Class		
SoundSystem	**Description:** The SoundSystem is responsible for alerting the driver if the vehicle detects an obstacle through audio of varying volume and frequency.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: None
	Attributes: volume : float	
	Operations: initiateWarning(float volume, float frequency) : boolean	

Table 31: Data dictionary entry for *SoundSystem*

Class		
Dashboard	**Description:** The Dashboard displays warning messages such as "Automatically Braking" notifies the driver of potential danger of mitigation actions from the system. If a sensor is occluded, then the dashboard also displays a message to prompt the user to clear the occlusion.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: None
	Attributes: message : String	
	Operations: displayWarning(String message) : boolean	

Table 32: Data dictionary entry for *Dashboard*

Class		
ToggleSystemButton	**Description:** The ToggleSystemButton is used to the system turn on or off.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: Is part of HumanMachineInterface
		Generalization: None
	Attributes: None	
	Operations: press() : void	

Table 33: Data dictionary entry for *ToggleSystemButton*

Class		
SprayNozzleButton	**Description:** The SprayNozzleButton is used to trigger the spray nozzle.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: Is part of HumanMachineInterface
		Generalization: None
	Attributes: None	
	Operations: press() : void	

Table 34: Data dictionary entry for *SprayNozzleButton*

Class		
WarningBox	**Description:** The WarningBox is used to the display messages on the HumanMachineInterface.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: Is part of HumanMachineInterface
		Generalization: None
	Attributes: message : String	
	Operations: displayWarning(String message) : boolean	

Table 35: Data dictionary entry for *WarningBox*

Class		
CameraFeed	**Description:** The CameraFeed is used to the display camera feeds on the HumanMachineInterface.	
	Export Control: Private	
	Relationships	**Associations:** None
		Aggregations: Is part of HumanMachineInterface
		Generalization: None
	Attributes: None	
	Operations: displayCameraFeed(Bitmap image) : boolean	

Table 36: Data dictionary entry for *CameraFeed*

Class		
BrakingSystem	**Description:** The BrakingSystem is responsible for deceleration. The braking system can receive signals to engage the brakes, and it will decrease the speed of the vehicle appropriately at the desired rate.	
	Export Control: Private	
	Relationships	**Associations:** Controller
		Aggregations: None
		Generalization: None
	Attributes: force : double	
	Operations: • applyBrakes(double force) : boolean	

Table 37: Data dictionary entry for *BrakingSystem*

4.3 Sequence Diagrams

Several sequence diagrams are introduced in this section, which each show a specific sequence of events that can happen in the BCAS system. In other words, each sequence diagram captures a scenario.

Each sequence diagram uses a specific notation. The blue boxes are instances of a class, and the dotted lines stretching down beneath them are their object lifelines. A solid arrow pointing from an object lifeline to another represents a message sent from one object to another. A dotted arrow represents a response to a received message. A bracketed statement above a message is a guard, which is a condition that that must be true for that message to be sent. A bracketed statement that begins with "for all" and is preceded by an asterisk denotes iteration, i.e., the message is sent more than once. A box labeled "opt" denotes a set of messages that will only be sent if the guard at the top of the box is true. A vertical arrow between two messages labeled with a time duration means that a certain amount of time must pass after the first message is sent before the second message can be sent.

4.3.1 Turning On the System

Figure 6 shows the sequence diagram describing how the system turns on. First, the driver shifts the gear switch. If the gear switch is in the reverse position, then the turnOn() method of the Controller is called. Next, the system invokes the turnOn() method of the sensors and enable() for the PedestrianWarningLight to turn on. If any of the sensors or the PedestrianWarningLight return false, then the controller determines there is a hardware failure and sets the hardware failure attribute to true. If a hardware failure is detected (i.e., hardwareFailure == true), then a warning message is displayed to the driver via the HumanMachineInterface and Dashboard by invoking the displayWarning(...) methods of the HumanMachineInterface and Dashboard.

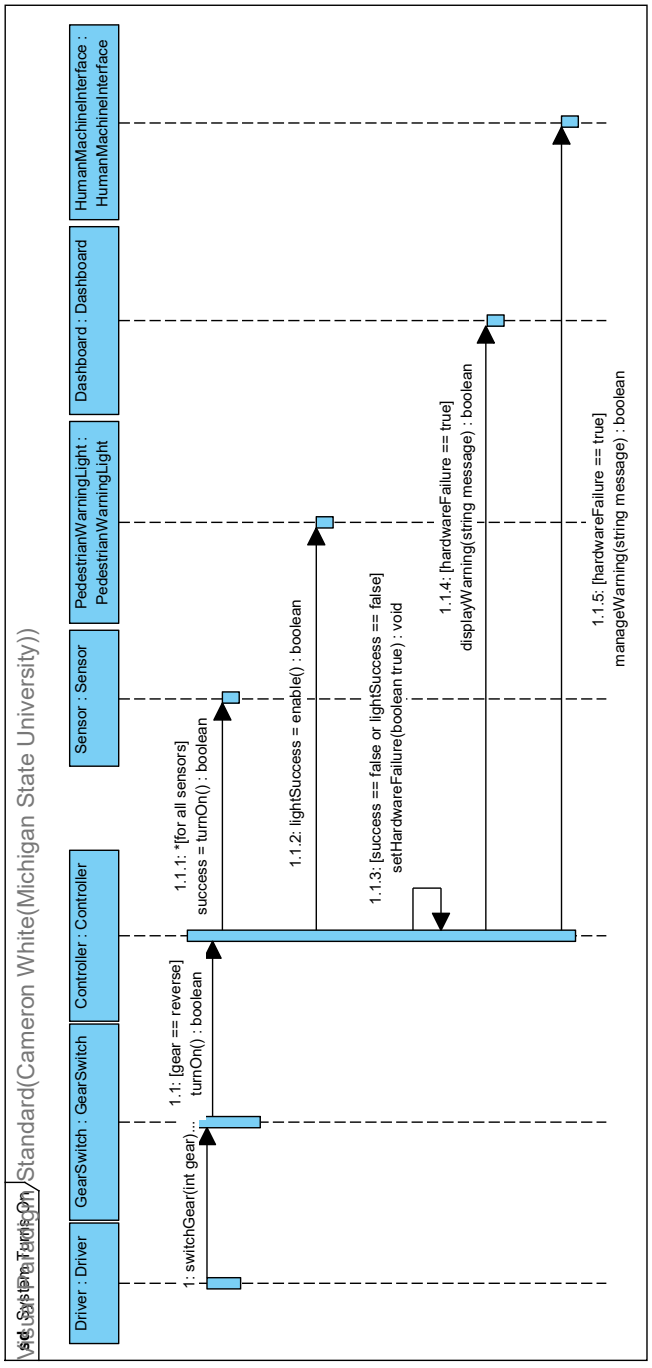

Figure 6: Sequence Diagram for Turning On the System

4.3.2 Object Detected On Camera

Figure 7 shows the sequence diagram for the Object Detected on Camera scenario. When the BCAS is active, the Controller obtains a camera feed from the CameraSensor. The Controller passes the data from the camera feed to the DetectionAlgorithm, which identifies and draws a thin box around the object or pedestrian in red and green, respectively. The modified camera feed is then returned to the Controller. The Controller then redirects the modified camera feed to the HumanMachineInterface to display to the driver through an InputStream.

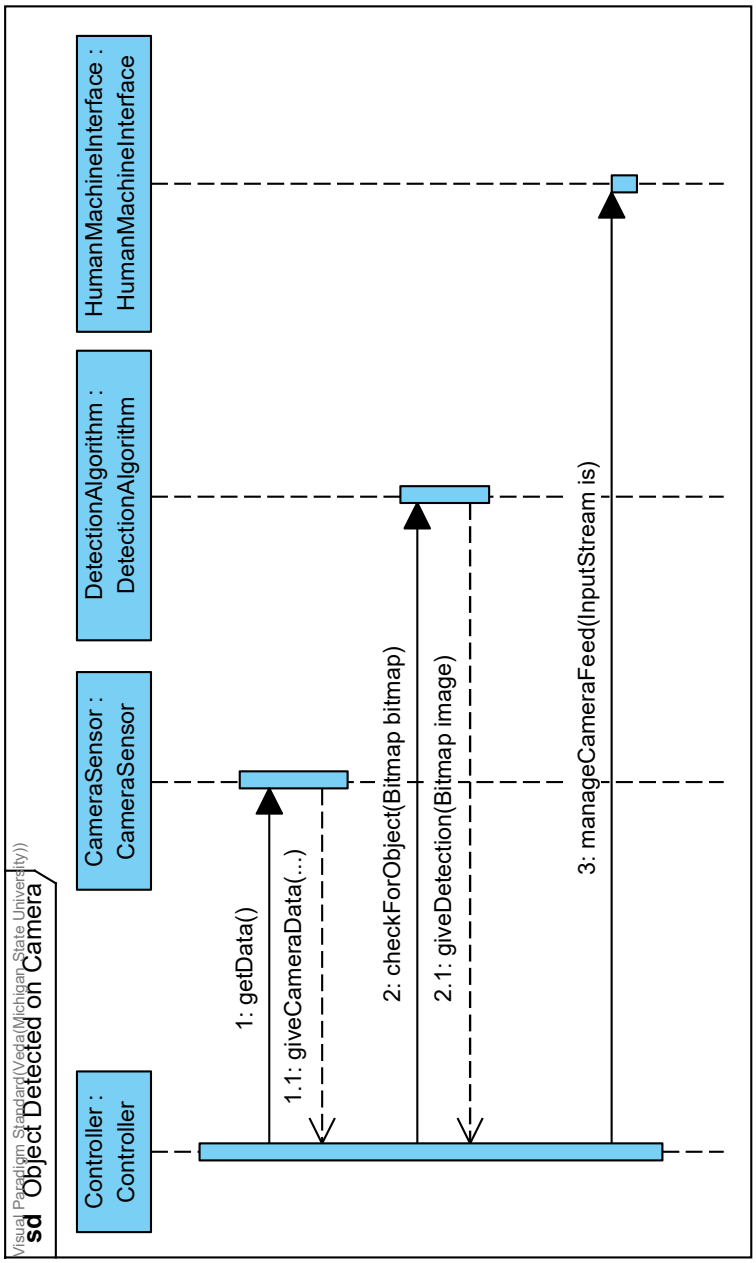

Figure 7: Sequence Diagram for Object Detected On Camera

4.3.3 System Override

Figure 8 shows the sequence diagram for the scenario where the driver presses the override button for the system. First, the driver presses the OverrideButton located on the steering wheel. The OverrideButton turns on or off its LED with toggleLED() and invokes the toggleOverride() method of the Controller. If the driver is overriding the system (i.e., state == suspended), then the Controller calls the disable() method for the PedestrianWarningLight. Alternatively, if the driver is enabling the system (i.e., state == enabled), then the Controller calls the enable() method for the PedestrianWarning-Light.

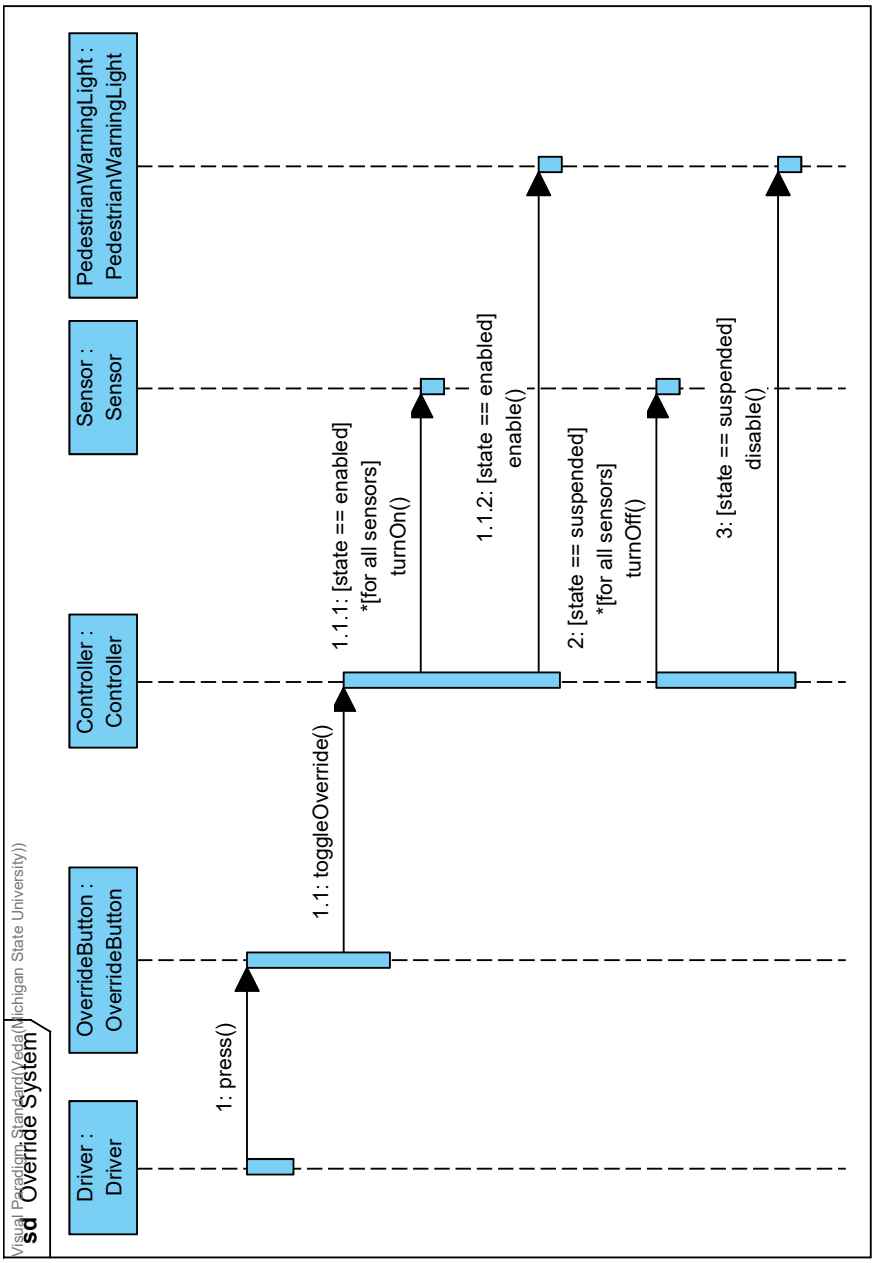

Figure 8: Sequence Diagram for a System Override

4.3.4 Disabling the System

Figure 9 shows the sequence diagram for the scenario where the driver disables the system from the HumanMachineInterface. The driver first uses the adjustSettings() method from the HumanMachineInterface to disable the system. The HumanMachineInterface invokes the disableSystem() method of the Controller, which consequently calls the disable() method of the PedestrianWarningLight.

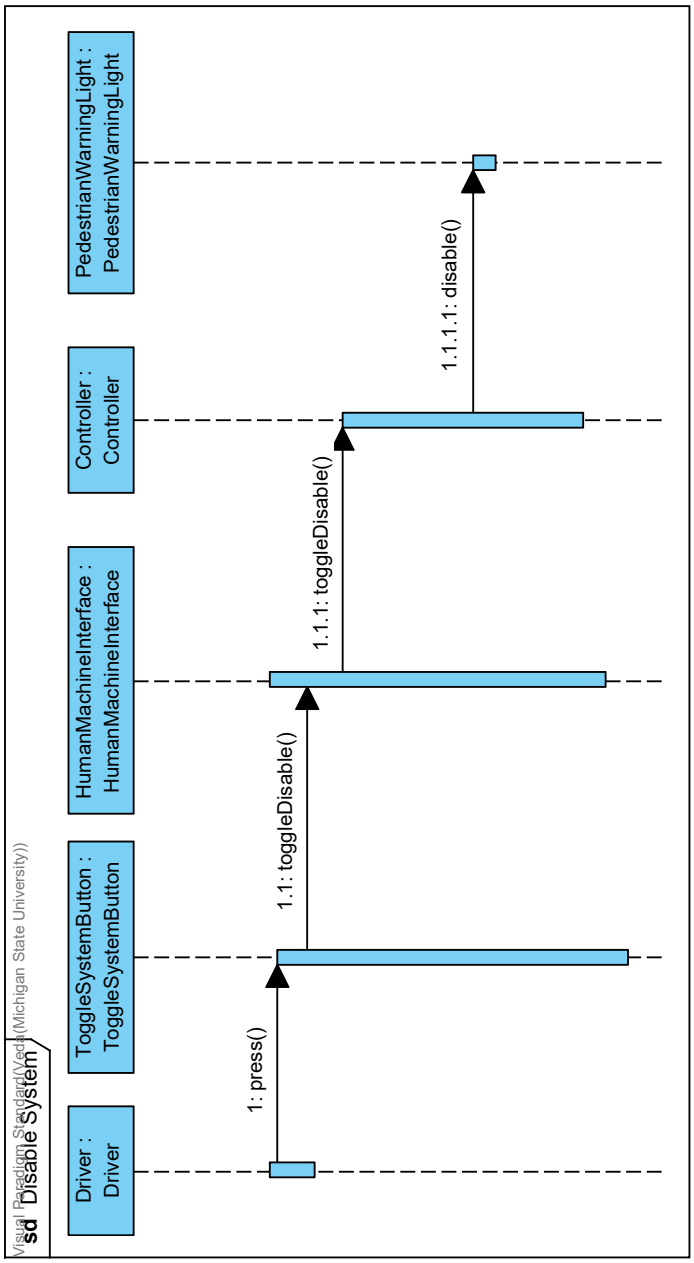

Figure 9: Sequence Diagram for Disabling the System

4.3.5 Minimize Impact After Collision Detected

Figure 10 shows the sequence diagram for the scenario where the BCAS detects a collision with an external variable and automatically brakes. First, the Controller uses the getData() methods of the CollisionSensor and the Accelerator to obtain the sensor values. The CollisionSensor returns a boolean of true if a collision is detected, and false otherwise. The Accelerator returns the current acceleration rate of the vehicle, which the Controller uses to detect unintended deceleration. If a collision has been detected (i.e., collisionDetected == true), then the controller automatically brakes, applying maximum braking force by calling the applyBrakes(...) method of the BrakingSystem. After 3 seconds of an automatic brake, the Controller calls releaseBrakes() to allow the driver to accelerate.

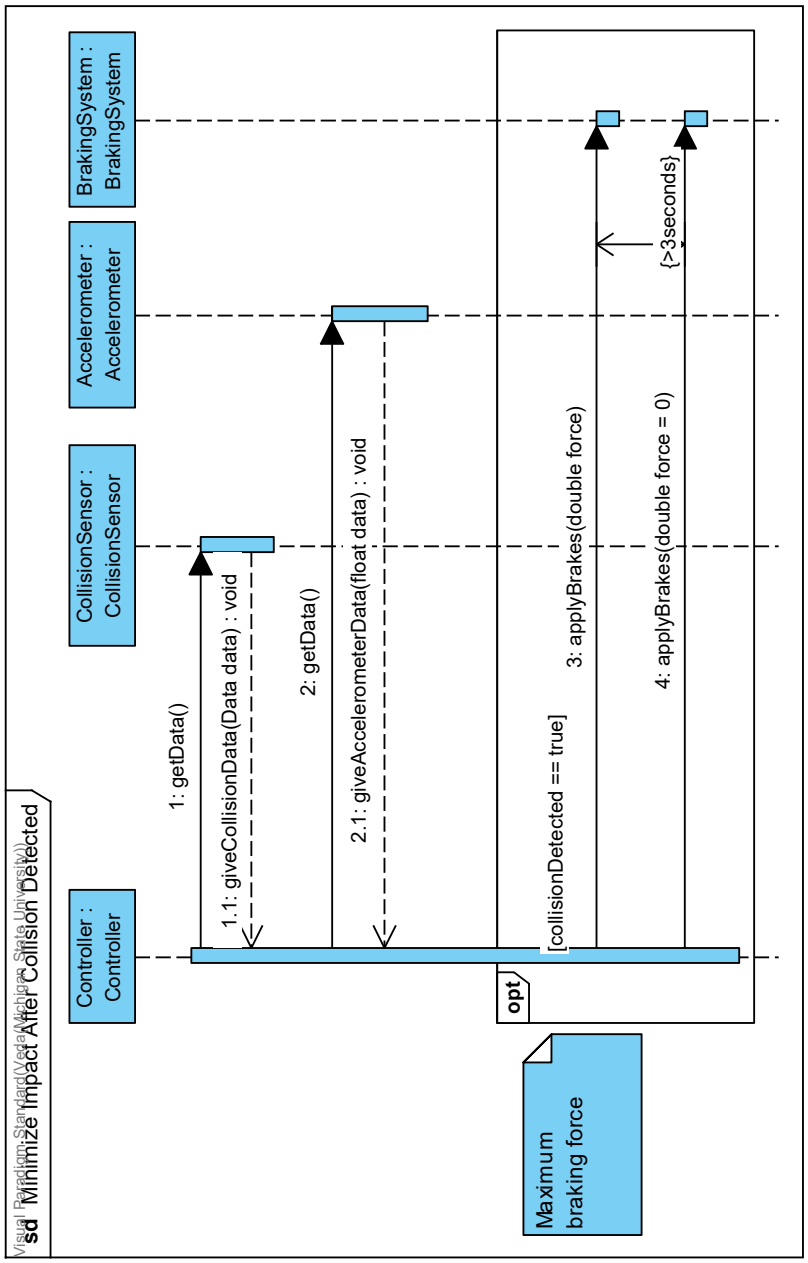

Figure 10: Sequence Diagram for Minimize Impact After Collision Detected

4.3.6 Object Detected Within 2 meters

Figure 11 shows the sequence diagram for the scenario where the BCAS detects an object or pedestrian within 2 meters. When the BCAS is active, the Controller obtains sensor values from the CameraSensor, LidarSensor, and UltrasonicSensor. The Controller uses the DetectionAlgorithm to detect objects or pedestrians using the camera feed and calculates the distance of the object or pedestrian using the data collected from the sensors. The handleIntervention() method is then invoked. If an object or pedestrian is detected within 2 meters of the vehicle (i.e., objectDistance $\leq$ 2m), then the Controller plays a periodic beep with initiateWarning(...). Next, the Controller invokes the applyBrakes(...) method to reduce the current speed by 25%. Both displayWarning(...) methods for the HumanMachineInterface and the Dashboard are used to display a message of "AUTOMATICALLY BRAKING" to the driver.

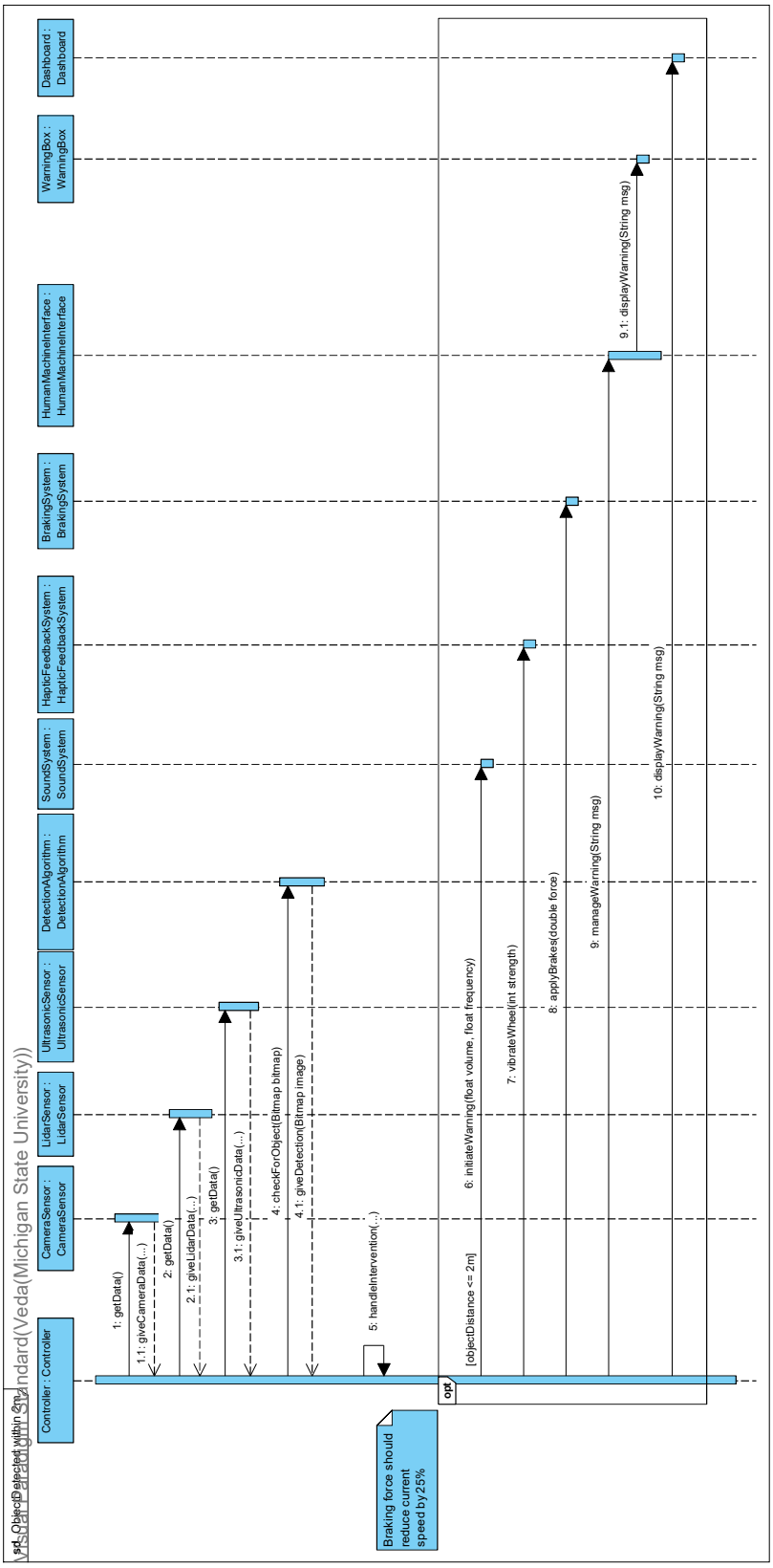

Figure 11: Sequence Diagram for Object Detected Within 2 meters

4.3.7 Imminent Collision

Figure 12 shows the sequence diagram for the scenario where the BCAS detects an imminent collision. When the BCAS is active, the Controller obtains sensor values from the CameraSensor, LidarSensor, and UltrasonicSensor. The Controller uses the DetectionAlgorithm to detect objects or pedestrians using the camera feed and calculates the distance of the object or pedestrian using the data collected from the sensors. If a collision is imminent (i.e., objectDistance $\leq$ 1m), then the Controller plays a periodic beep with initiateWarning(...). Next, the Controller invokes the applyBrakes(...) method to apply the maximum braking force to stop the vehicle. Both displayWarning(...) methods for the HumanMachineInterface and the Dashboard are used to display a message of "AUTOMATICALLY BRAKING" to the driver. The Controller then provides haptic feedback for the driver by invoking the vibrateWheel(...) method. Finally, three seconds after the brakes have been applied, the Controller calls the releaseBrakes() method to allow the driver to move the vehicle.

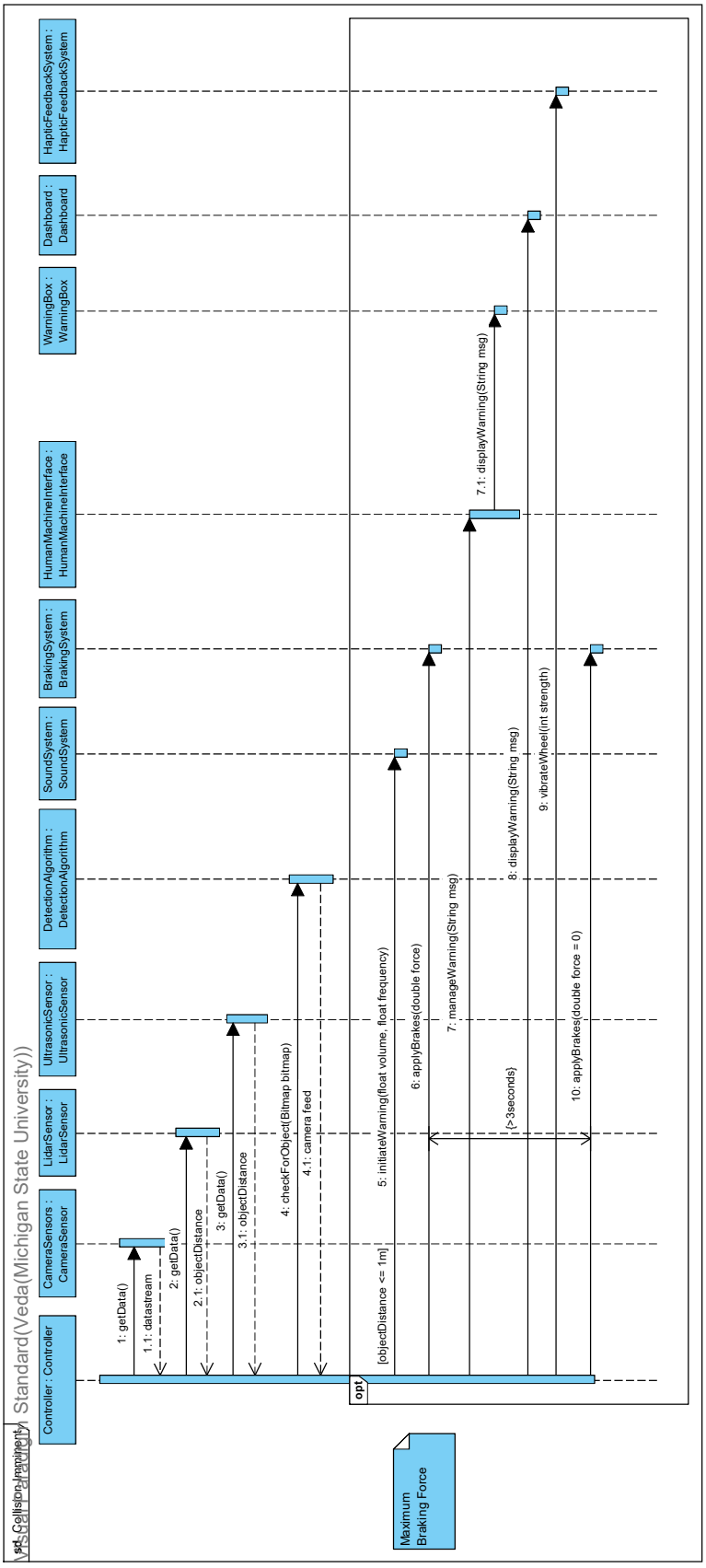

Figure 12: Sequence Diagram for an Imminent Collision

4.4 State Diagrams

This section introduces the state diagrams used to model all the possible states and shows the behavior of one object over many use cases.

In a state diagram, the beginning state of the diagram is depicted by a black-filled circle with an arrow leading to a state. States of the system are represented by blue boxes. Arrows that connect two states represent state transitions, with the corresponding descriptions or guards attached to the arrow.

4.4.1 Accelerometer

Figure 13 depicts the state diagram for the accelerometer. The Accelerometer's initial and only state is the Monitoring state. While in the Monitoring state, the Controller can request a measurement via the Accelerometer's getData() operation where the Accelerator responds with sending over the data by calling the Controller's giveAccelerometerData() operation.

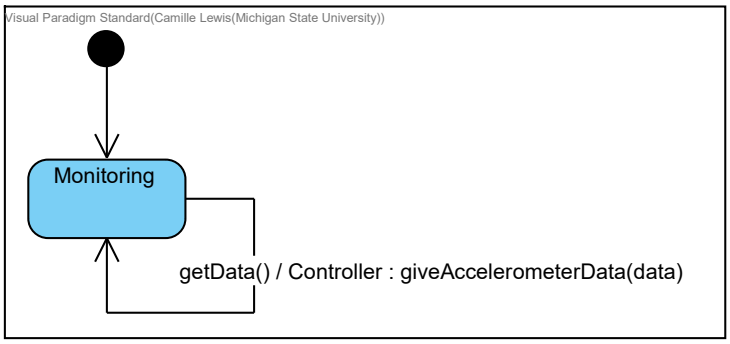

Figure 13: State diagram for the *Accelerometer*

4.4.2 Braking System

Figure 14 depicts the state diagram for the braking system. The BrakingSystem class has two states: Not Braking and Braking. BrakingSystem begins in the Not Braking state. In either state, the BrakingSystem class can receive an applyBrakes() message from the Controller. The force passed as a parameter in the operation call determines if the BrakingSystem changes state. If the BrakingSystem is in the Not Braking state and receives a message with a force value greater than zero, then the BrakingSystem transitions to the Braking state. If the BrakingSystem is in the Braking state and receives a message with a force value equal to zero, then the BrakingSystem transitions to the Not Braking state.

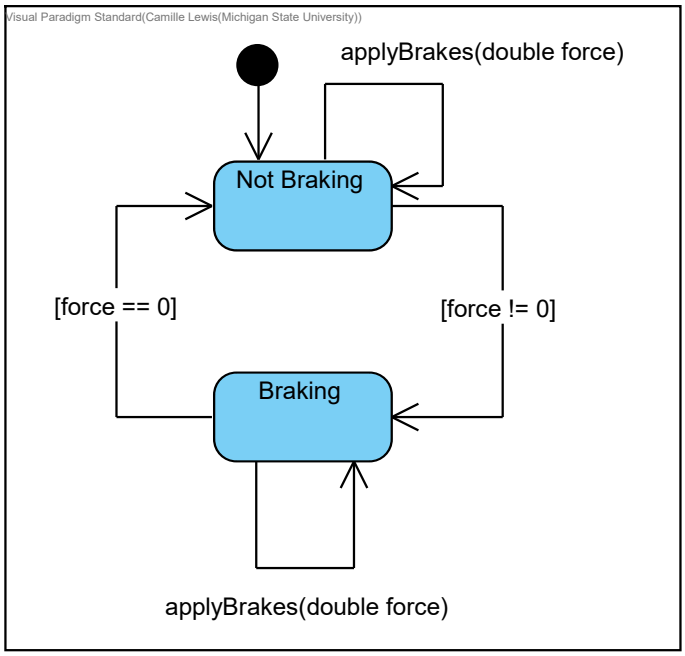

Figure 14: State diagram for the *Braking System*

4.4.3 Dashboard

Figure 15 depicts the state diagram for the dashboard. The Dashboard class has two states: Not Displaying and Displaying. Dashboard begins in the Not Displaying state. In either state, the Dashboard class can receive a displayWarning() call from the Controller. The string parameter message determines if the Dashboard changes state. If the Dashboard is in the Not Displaying state and receives a message with a non-empty string, then the Dashboard transitions to the Displaying state. If the Dashboard is in the Displaying state and receives a message with an empty string, then the Dashboard transitions to the Not Displaying state.

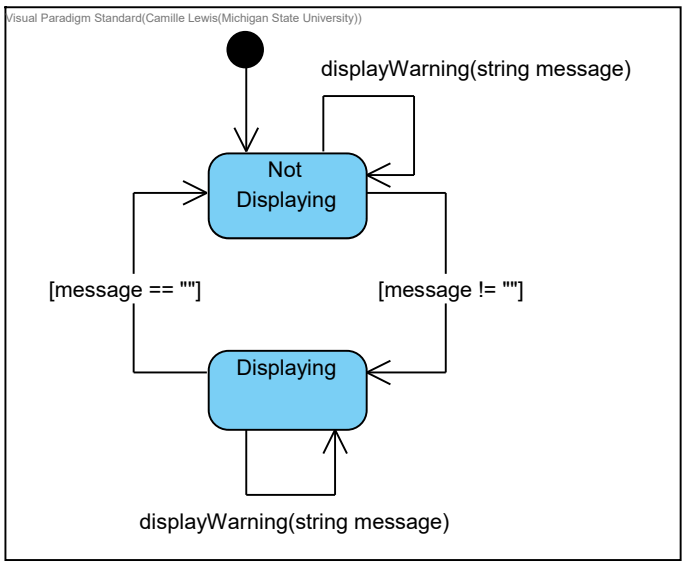

Figure 15: State diagram for the *Dashboard*

4.4.4 Gear Switch

Figure 16 depicts the state diagram for the gear switch. The Gear Switch class comprises of the Park and Reverse state. The Gear Switch class begins with the vehicle in Park. When the switchGear(int gear) method is called, the Gear Switch class transitions from park to reverse and the vehicle begins the reverse operation. If the state is in reverse and the switchGear(int gear) function is called, the state transitions back to Park.

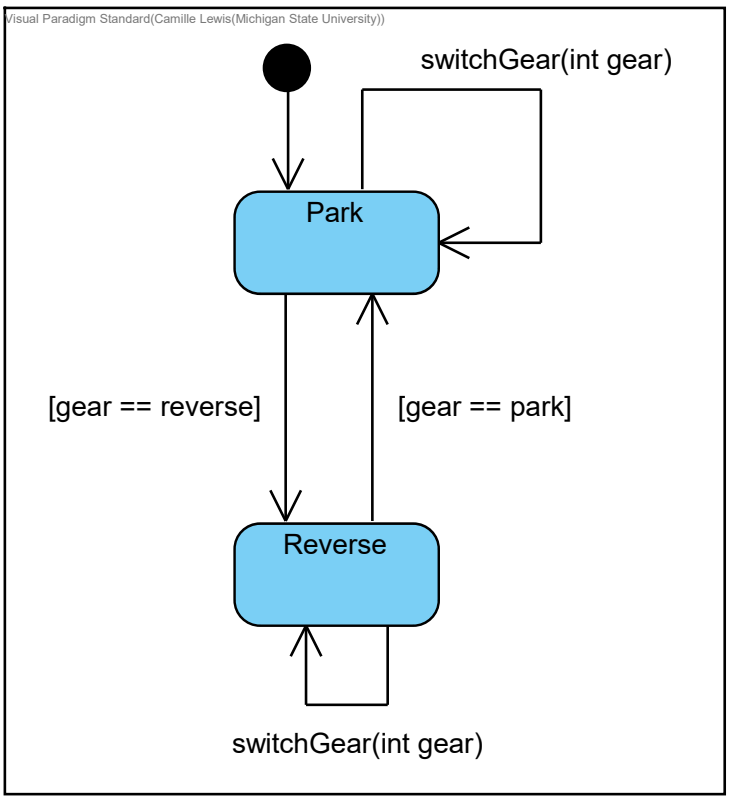

Figure 16: State diagram for the *GearSwitch*

4.4.5 Haptic Feedback System

Figure 17 depicts the state diagram for the Haptic Feedback System. The HapticFeedbackSystem begins in the Not Vibrating state, where it can receive a vibrateWheel(int strength) operation call from the Controller. If the strength value is nonzero, then the HapticFeedbackSystem transitions to the Vibrating state. When in the Vibrating state, the HapticFeedbackSystem can receive a vibrateWheel(int strength) operation call from the Controller. While in this state, if the stength value is equal to zero, then the HapticFeedbackSystem transitions to the Not Vibrating state.

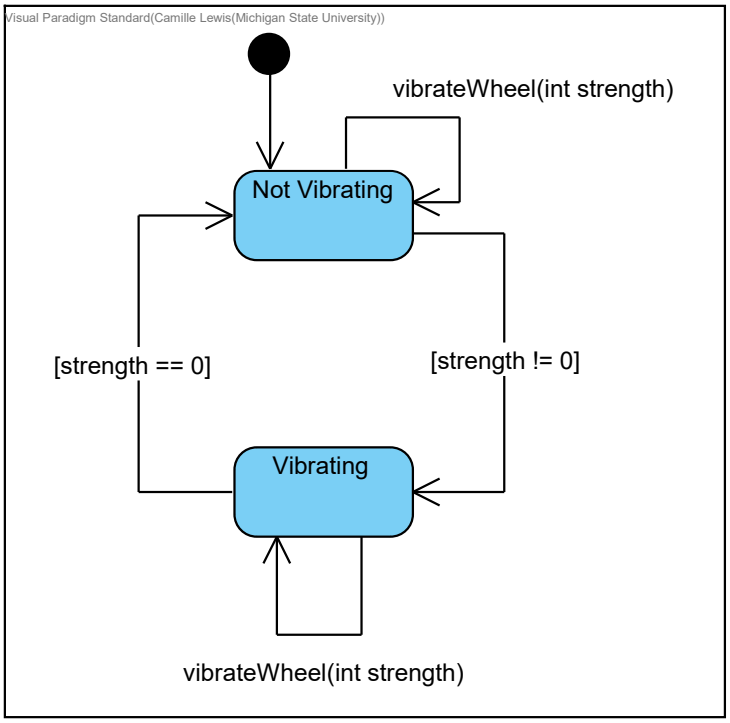

Figure 17: State diagram for the *Haptic Feedback System*

4.4.6 Sound System

Figure 18 depicts the state diagram for the Sound System. The SoundSystem class begins in the Not Warning state. The SoundSystem can receive initiateWarning(float volume, float frequency) calls from the Controller when in any state. While in the Not Warning state, if the volume value is nonzero, then SoundSystem transitions to the Sound Warning state. While in the Sound Warning state, if the volume value is equal to zero, then SoundSystem transitions to the Not Warning state.

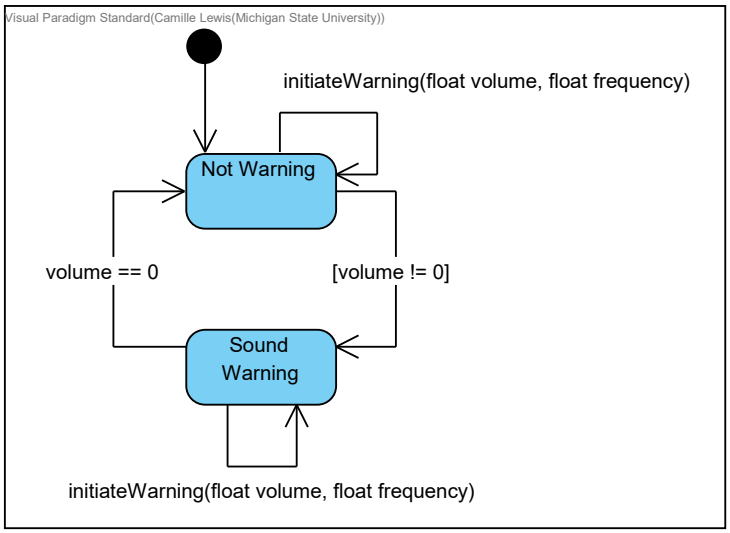

Figure 18: State diagram for the *Sound System*

4.4.7 Warning Box System

Figure 19 depicts the state diagram for the Warning Box System. The WarningBox class has two states: Not Displaying and Displaying. WarningBox begins in the Not Displaying state. In either state, the WarningBox class can receive a displayWarning(string message) call from the Controller. The string parameter message determines if the WarningBox changes state. If the WarningBox is in the Not Displaying state and receives a message with a non-empty string, then the WarningBox transitions to the Displaying state. If the WarningBox is in the Displaying state and receives a message with an empty string, then the WarningBox transitions to the Not Displaying state.

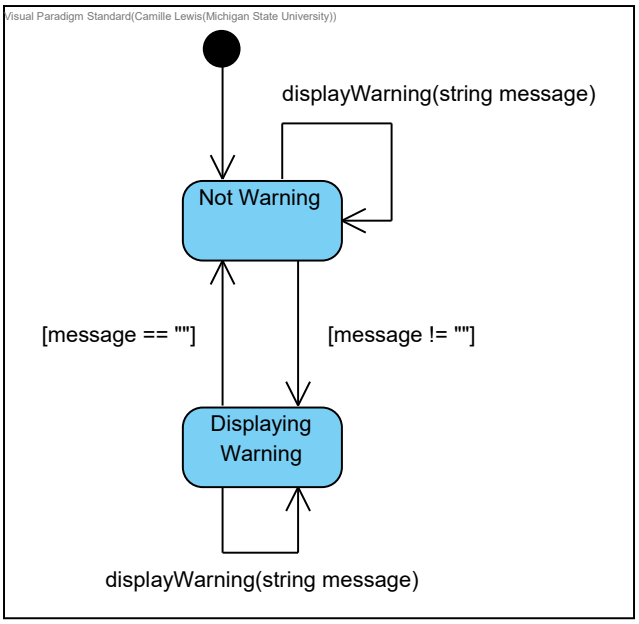

Figure 19: State diagram for the *Warning Box System*

4.4.8 Camera Feed

Figure 20 depicts the state diagram for the camera feed. The CameraFeed class has one state: Displaying Feed, where it can receive displayCameraFeed(Bitmap image) operation calls from the HumanMachineInterface and display the image in the UI element.

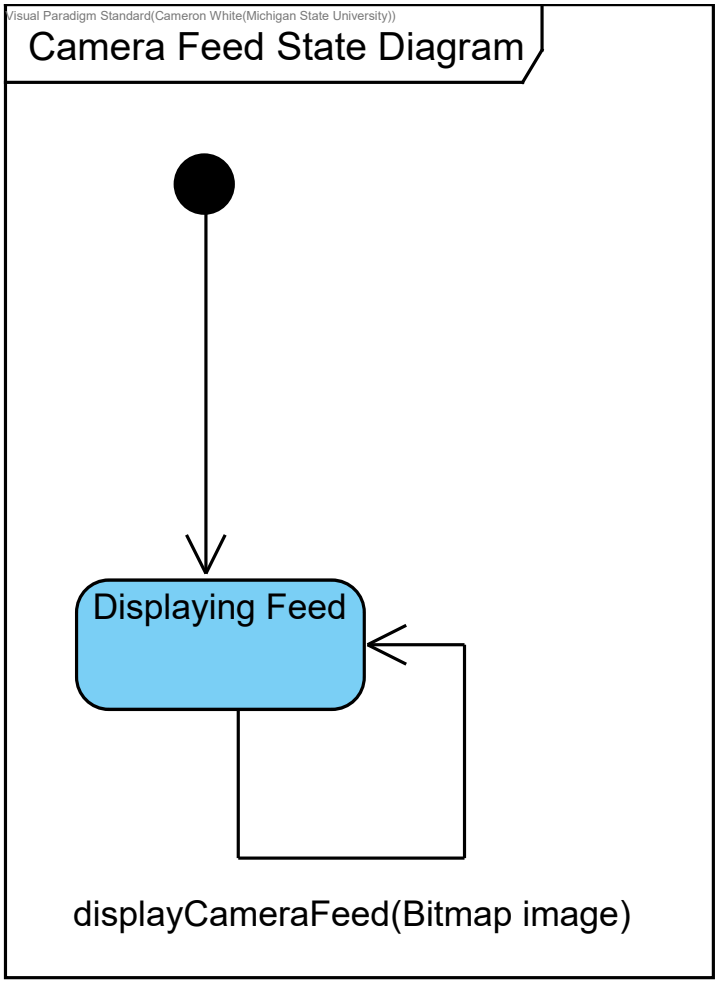

Figure 20: State diagram for the *Camera Feed*

4.4.9 Camera Sensor

Figure 21 depicts the state diagram for the camera sensor. The CameraSensor begins in the Off state. Receiving a turnOn() operation call from the Controller causes the CameraSensor to transition to the Getting Data state. A turnOff() operation call from the Controller while in the Getting Data state causes the CameraSensor to transition back to the Off state. While in the Getting Data state, the CameraSensor can receive a getData() request from the Controller, and the CameraSensor responds by sending the requested data using the Controller's giveCameraData(Bitmap image, string id) operation.

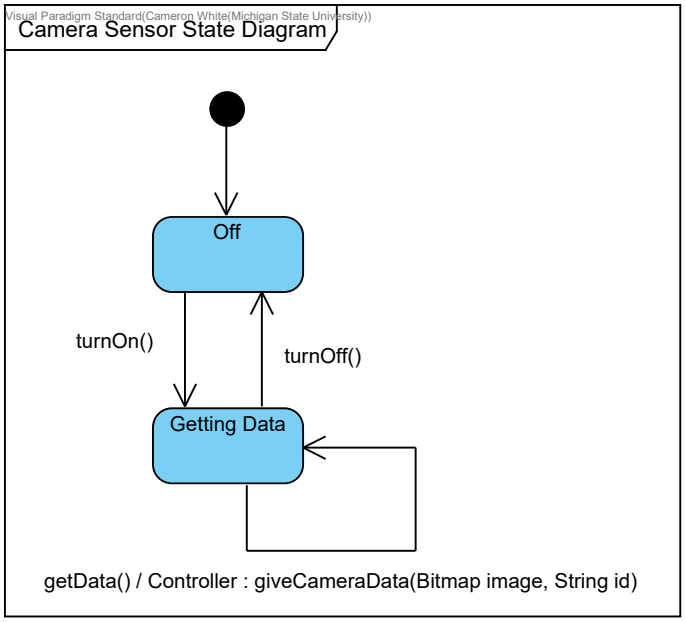

Figure 21: State diagram for the *Camera Sensor*

4.4.10 Collision Sensor

Figure 22 depicts the state diagram for the collision sensor. The CollisionSensor begins in the Off state. Receiving a turnOn() operation call from the Controller causes the CollisionSensor to transition to the Getting Data state. A turnOff() operation call from the Controller while in the Getting Data state causes the CollisionSensor to transition back to the Off state. While in the Getting Data state, the CollisionSensor can receive a getData() request from the Controller, and the CollisionSensor responds by sending the requested data using the Controller's giveCollisionData(Data data) operation.

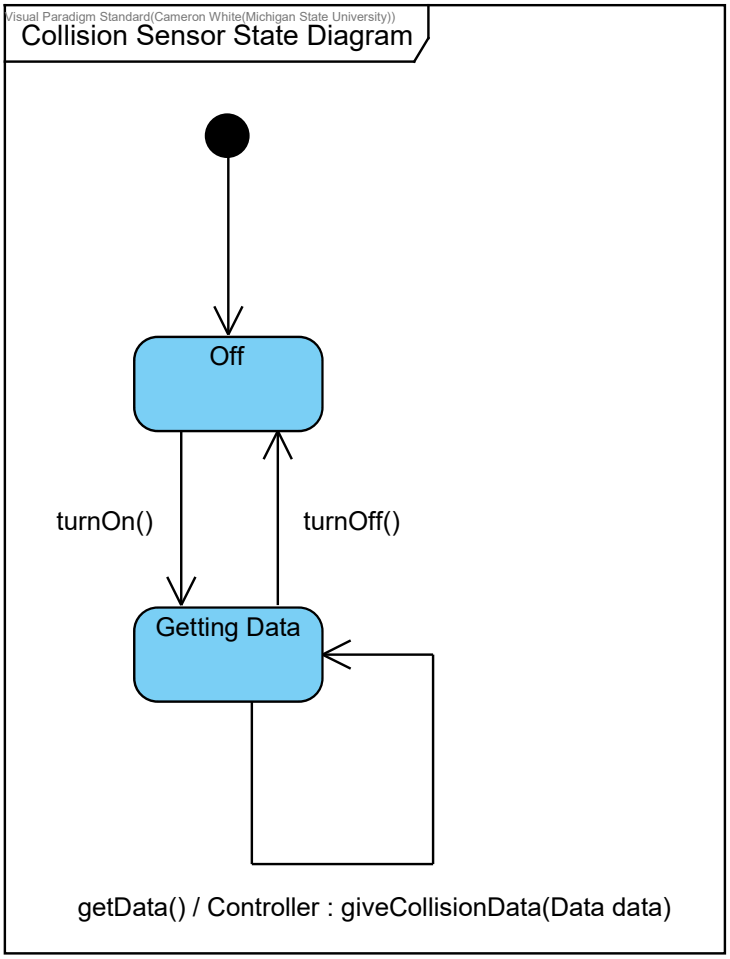

Figure 22: State diagram for the *Collision Sensor*

4.4.11 Detection Algorithm

Figure 23 depicts the state diagram for the Detection Algorithm. The Detection-Algorithm has one state, the Idle state, where it can receive a checkForObject(Bitmap image) operation call from the Controller, and the DetectionAlgorithm will respond by processing the image and sending a modified image back via the Controller's giveDetection(Bitmap image) operation.

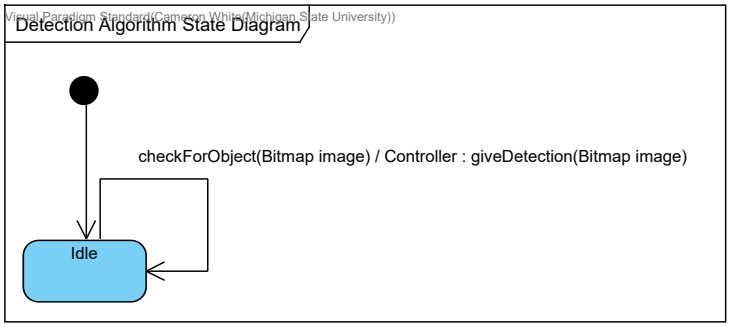

Figure 23: State diagram for the *Detection Algorithm*

4.4.12 Heated Compartment

Figure 24 depicts the state diagram for the heated compartment. The HeatedCompartment begins in the Off state. While in the Off state, the HeatedCompartment can receive a turnOn() operation call from the VehiclePowerButton, and this causes the HeatedCompartment to transition to the On state. Withing the On state, there are two concurrent state machines: Check Temp and Manage Heat. In the Check Temp concurrent state machine, the only state is the Checking Temp state where the HeatedCompartment calls its own checkTemp() operation every 10 seconds, which sets the currentTemp attribute. In the Manage Heat concurrent state machine, the initial state is the Not Heating state. When in the Not Heating state, if the currentTemp attribute value is less than the threshold temperature defined in the system, then the HeatedCompartment transitions to the Heating state. When in the Heating state, if the currentTemp is greater than or equal to the threshold temperature, then the HeatedCompartment transitions to the Not Heating state.

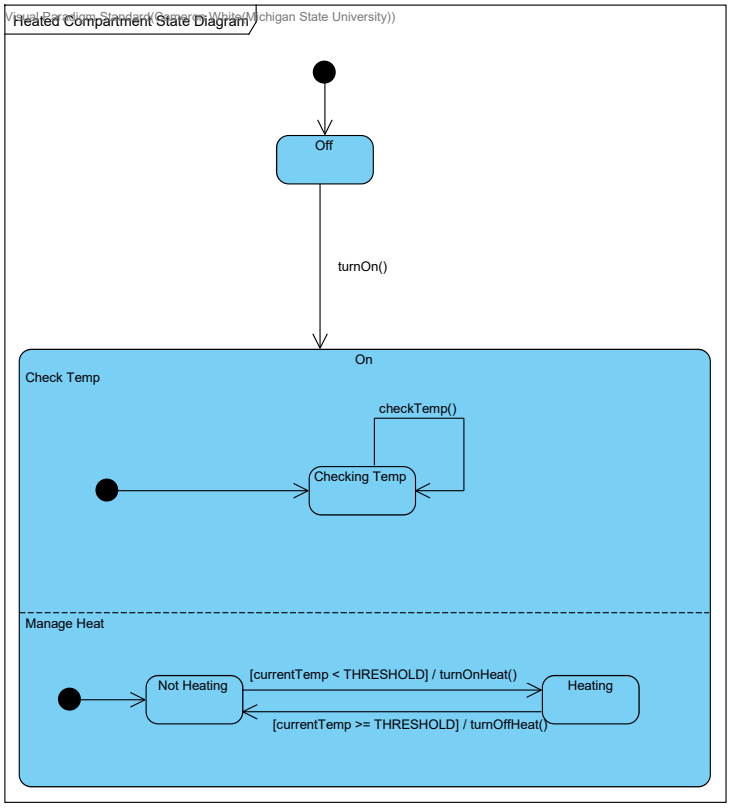

Figure 24: State diagram for the *Heated Compartment*

4.4.13 Human Machine Interface

Figure 25 depicts the state diagram for the Human Machine Interface (HMI). The HMI begins in the Idle state. In this state, it can receive toggleDisable() messages from the ToggleSystemButton, and the HMI will respond by calling the Controller's toggleDisable(). While in the Idle state, the HMI can also receive toggleSystemOn() messages from the Controller, and the HMI will set its systemOn attribute. When in the Idle state, if the systemOn attribute value is true, then the HMI transitions to the BCAS On state. While in the BCAS On state, the HMI can also receive toggleSystemOn() calls from the Controller. When in the BCAS On state, if the systemOn attribute value is false, then the HMI transitions to the Idle state. The BCAS On state is composed of 3 concurrent state machines: Camera Feed, Warning, and Spray Nozzle. The Camera Feed concurrent state machine has one state, Displaying Feed, where the HMI can receive manageCameraFeed(Bitmap image) messages from the Controller, which causes the HMI to send the feed to the CameraFeed class via displayCameraFeed(Bitmap image). The Warning concurrent state machine has one state: Displaying Warning. In this state, the HMI can receive manageWarning(string message) calls from the Controller, and the HMI will send the message on to the WarningBox via displayWarning(string message). The third and final concurrent state machine in the BCAS On state is Spray Nozzle. This state machine's sole state is Activating Spray Nozzle. In this state, the HMI can receive activateSprayNozzle() messages from the SprayNozzleButton, and calls the engageSprayNozzle() function to activate the spray nozzle. After that, the state remains in Activating Spray Nozzle.

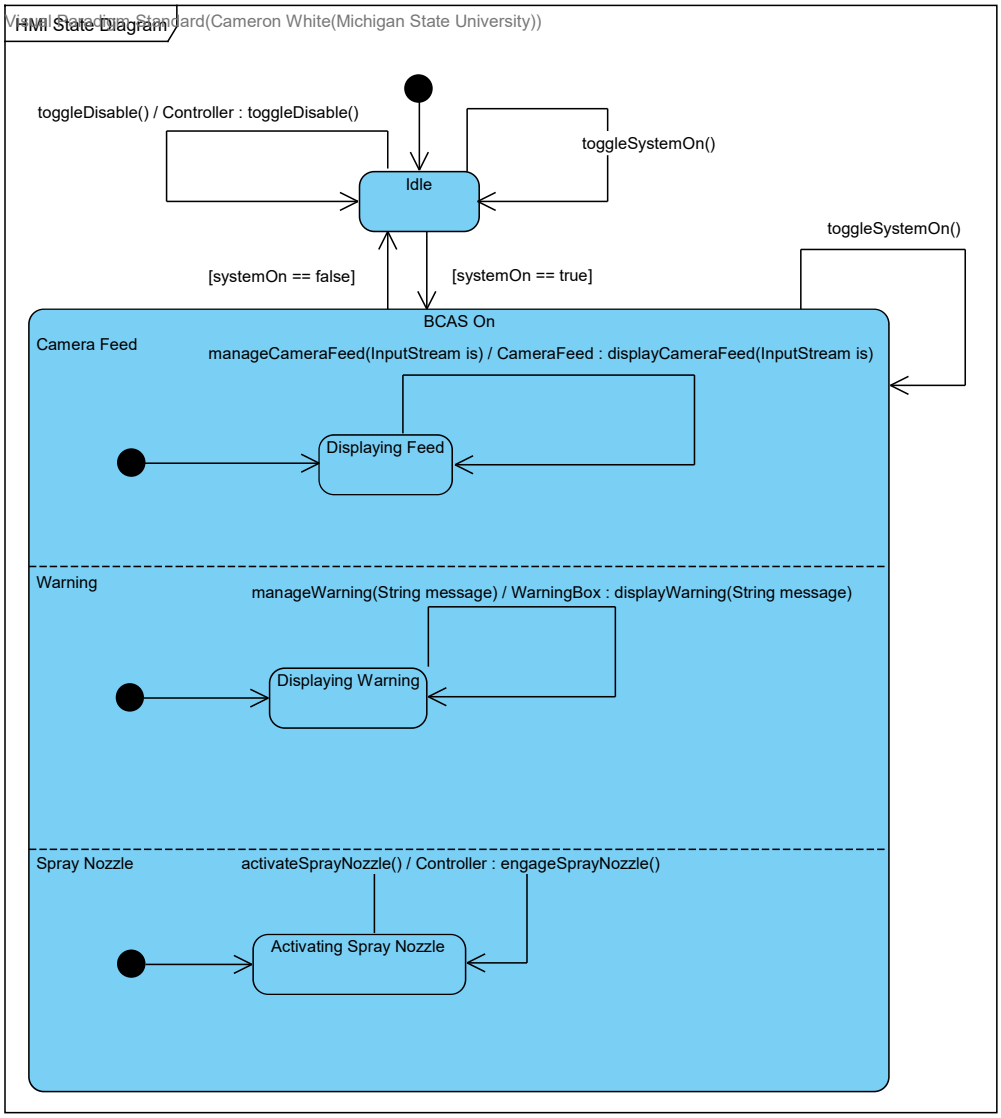

Figure 25: State diagram for the *HMI*

4.4.14 Lidar Sensor

Figure 26 depicts the state diagram for the lidar sensor. The lidar sensor begins in the Off state. When it receives the turnOn() message from the Controller, the lidar sensor enters the Getting Data state. In the Getting Data state, the lidar sensor repeatedly provides the Controller with lidar sensor data by calling the giveLidarData(Data data) function of the Controller class. When the lidar sensor receives a turnOff() message from the Controller, the state transitions to Off.

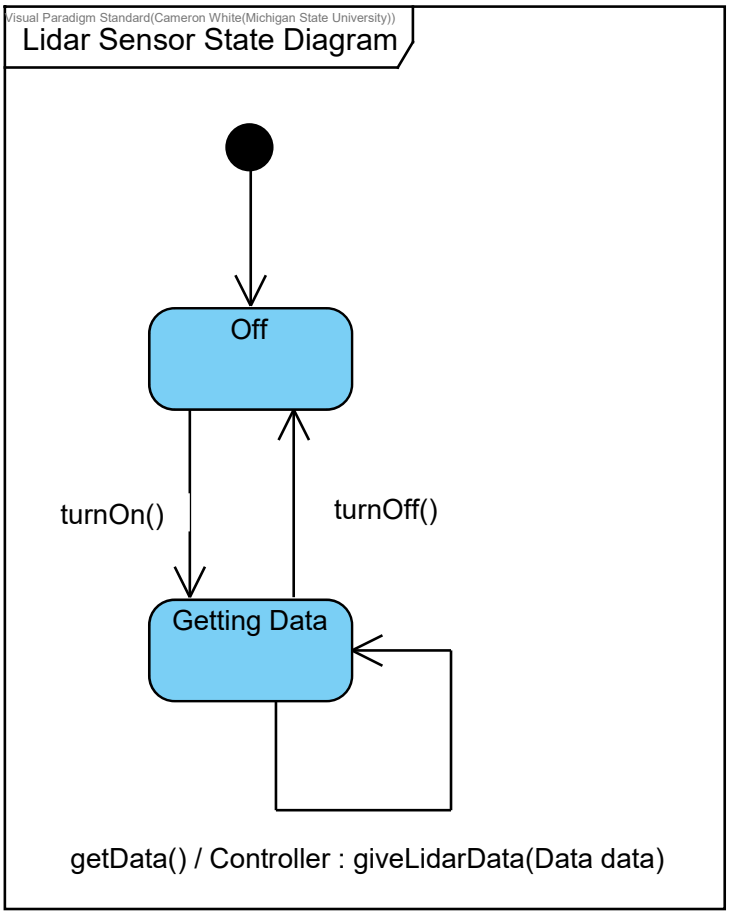

Figure 26: State diagram for the *Lidar Sensor*

4.4.15 Override Button

Figure 27 depicts the state diagram for the override button. The override button begins in the Idle state. When the button is pressed, the toggleOverride() function of the Controller is triggered. The state of the override button returns to Idle after the message.

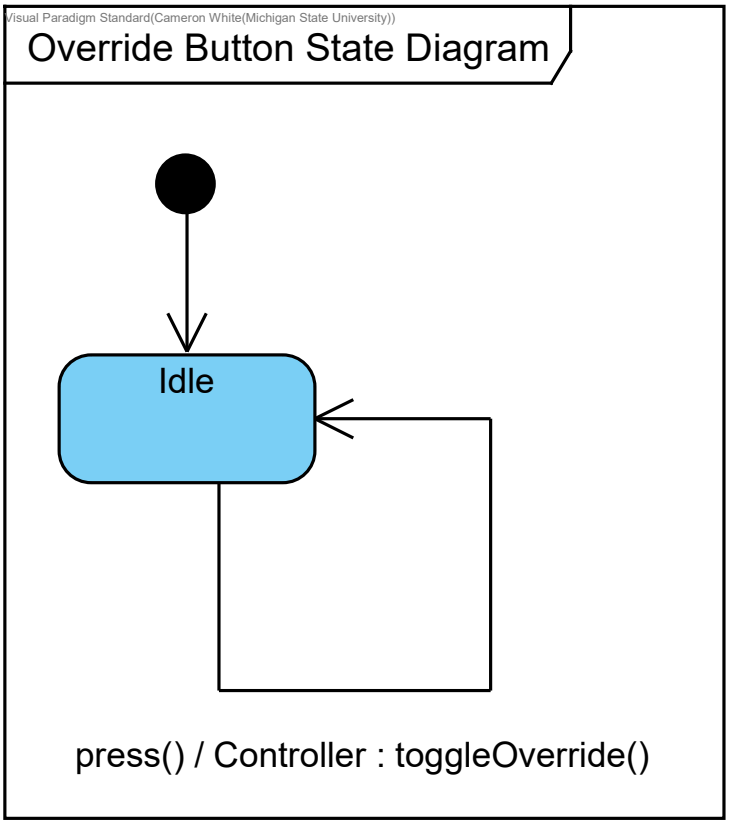

Figure 27: State diagram for the *Override Button*

4.4.16 Pedestrian Warning Light

Figure 28 depicts the state diagram for the pedestrian warning light. The pedestrian warning light begins in the Off state. When it receives the enable() message from the Controller, it transitions the state to On. When it receives the disable() message while the state is in On, the state returns to Off.

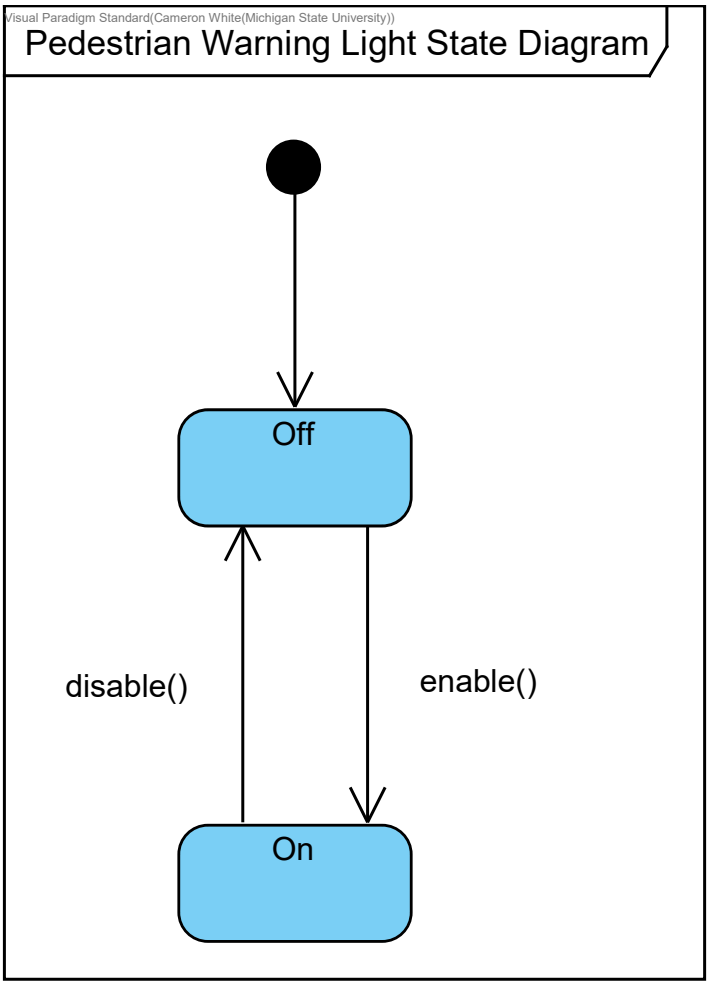

Figure 28: State diagram for the *Pedestrian Warning Light*

4.4.17 Spray Nozzle

Figure 29 depicts the state diagram for the spray nozzle. The spray nozzle begins in the Idle state. When it receives the turnOn() message from the Controller, it transitions the state to Spraying. When it receives the turnOff() message while the state is in Spraying, the state returns to Idle.

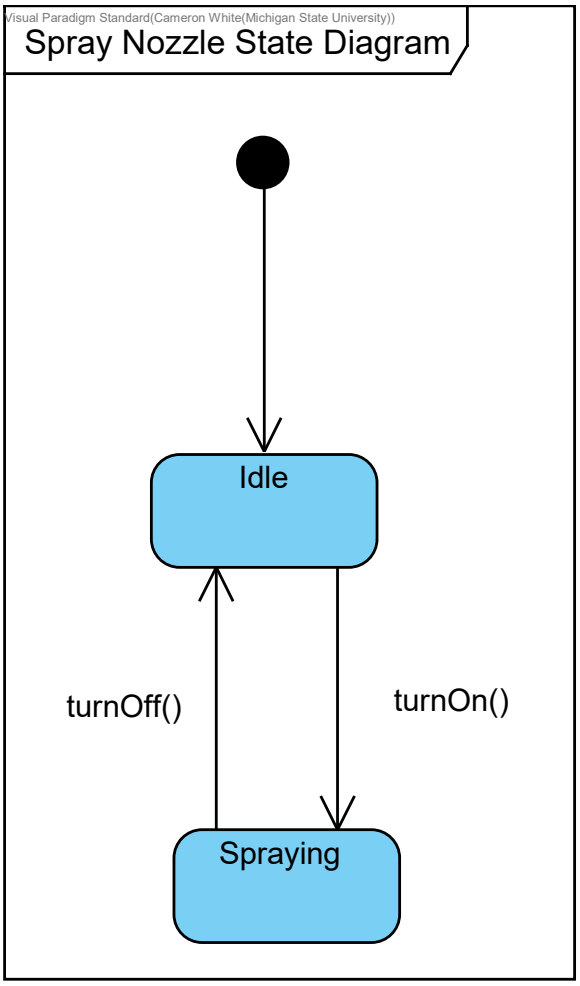

Figure 29: State diagram for the *spray nozzle*

4.4.18 Spray Nozzle Button

Figure 30 depicts the state diagram for the spray nozzle button. The spray nozzle button state begins in the Idle state. When the button is pressed, the activateSprayNozzle() function of the HumanMachineInterface is called. The state then returns to Idle.

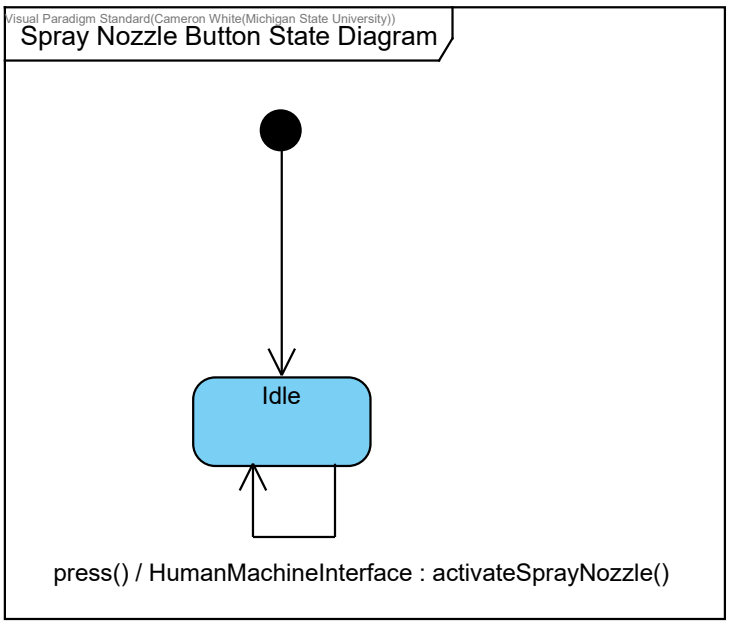

Figure 30: State diagram for the *Spray Nozzle Button*

4.4.19 Toggle System Button

Figure 31 depicts the state diagram for the toggle system button. The toggle system button state begins in the Idle state. When the button is pressed, the toggleSystemOn() function of the HumanMachineInterface is called. The state then returns to Idle.

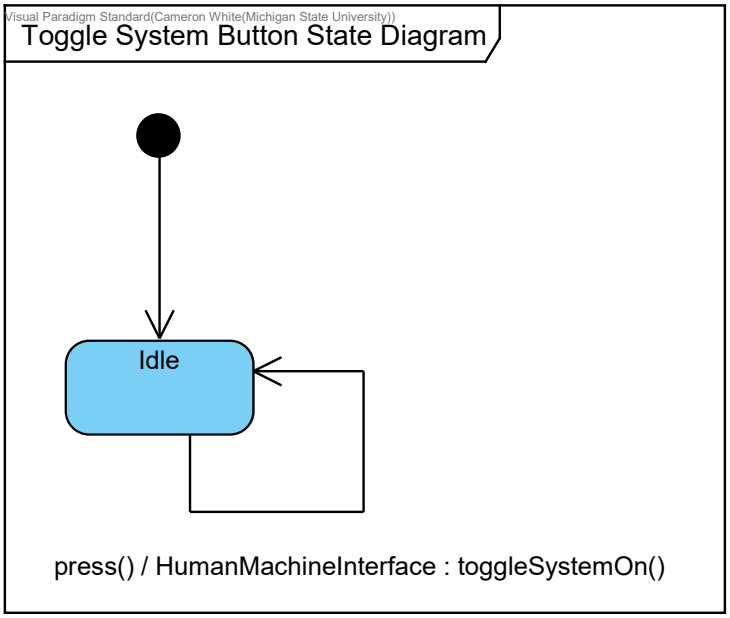

Figure 31: State diagram for the *Toggle System Button*

4.4.20 Ultrasonic Sensor

Figure 32 depicts the state diagram for the ultrasonic sensor. The ultrasonice sensor begins in the Off state. When it receives the turnOn() message from the Controller, it enters the Getting Data state. In this state, the ultrasonic sensor obtains sensor data and provides it to the Controller by calling the giveUltrasonicData(Data data) function. The state then returns to Getting Data. When the ultrasonic sensor receives the turnOff() message in the Getting Data state from the Controller, it returns to the Off state.

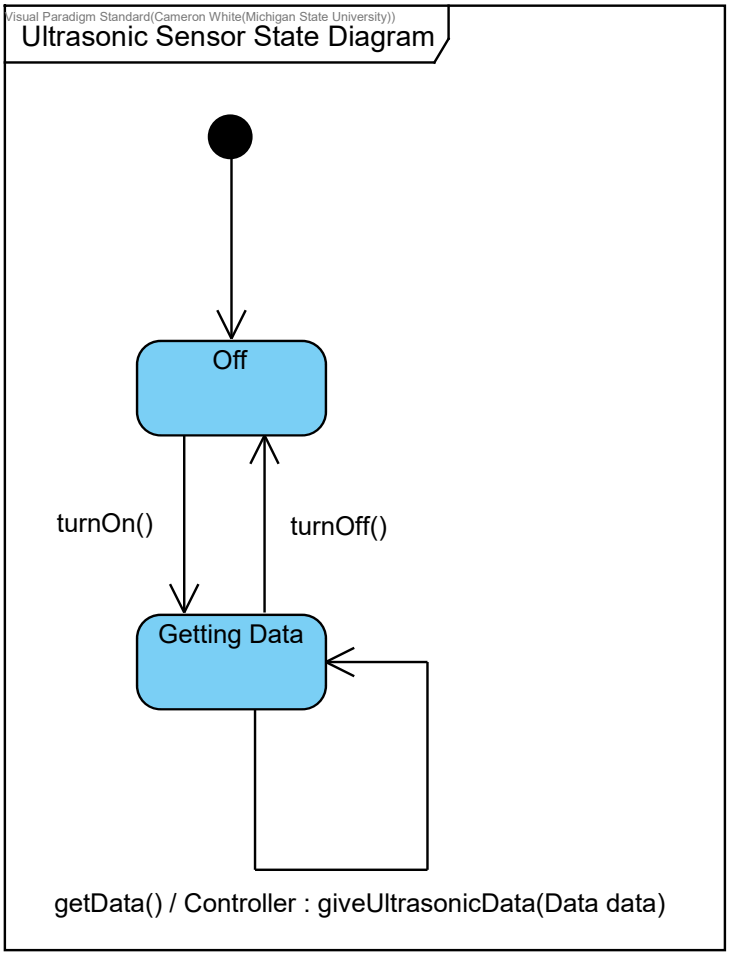

Figure 32: State diagram for the *Ultrasonic Sensor*

4.4.21 Controller

Figure 33 depicts the state diagram for the controller. The initial state of the Controller is BCAS Off. In this state, the Controller can accept operation calls from the HumanMachineInterface to toggle whether the system is disabled or enabled. A turnOn() operation call causes the Controller to transition to the Configuration state. In this state, all of the sensors in the system and the PedestrianWarningLight are turned on. If any errors occur during boot up, then the setHardwareFailure() operation is invoked and the Controller transitions to the BCAS On state. If no issues are detected, then the Controller transitions to the BCAS On state without any actions. In the BCAS On state, there are 4 concurrent state machines: Data Collection, Determine Decision, Toggle Override, and Spray Nozzle. In the Data Collection state machine, the Controller requests data from all sensors, sends received camera data to the DetectionAlgorithm, and sends modified camera data to the HumanMachineInterface. In the Determine Decision state machine, the Controller modifies the warning messages and braking force appropriately depending on the system state and the current minimum object distance measurement. In the Toggle Override state machine, the Controller can receive toggleOverride() calls from the OverrideButton. In the SprayNozzle state machine, the Controller handles messages from the HumanMachineInterface and sends messages to the SprayNozzle to start and stop spraying. While in the BCAS On state, the Controller can receive a turnOff() message; in response, the Controller turns off the necessary components and transitions back to the BCAS Off state.

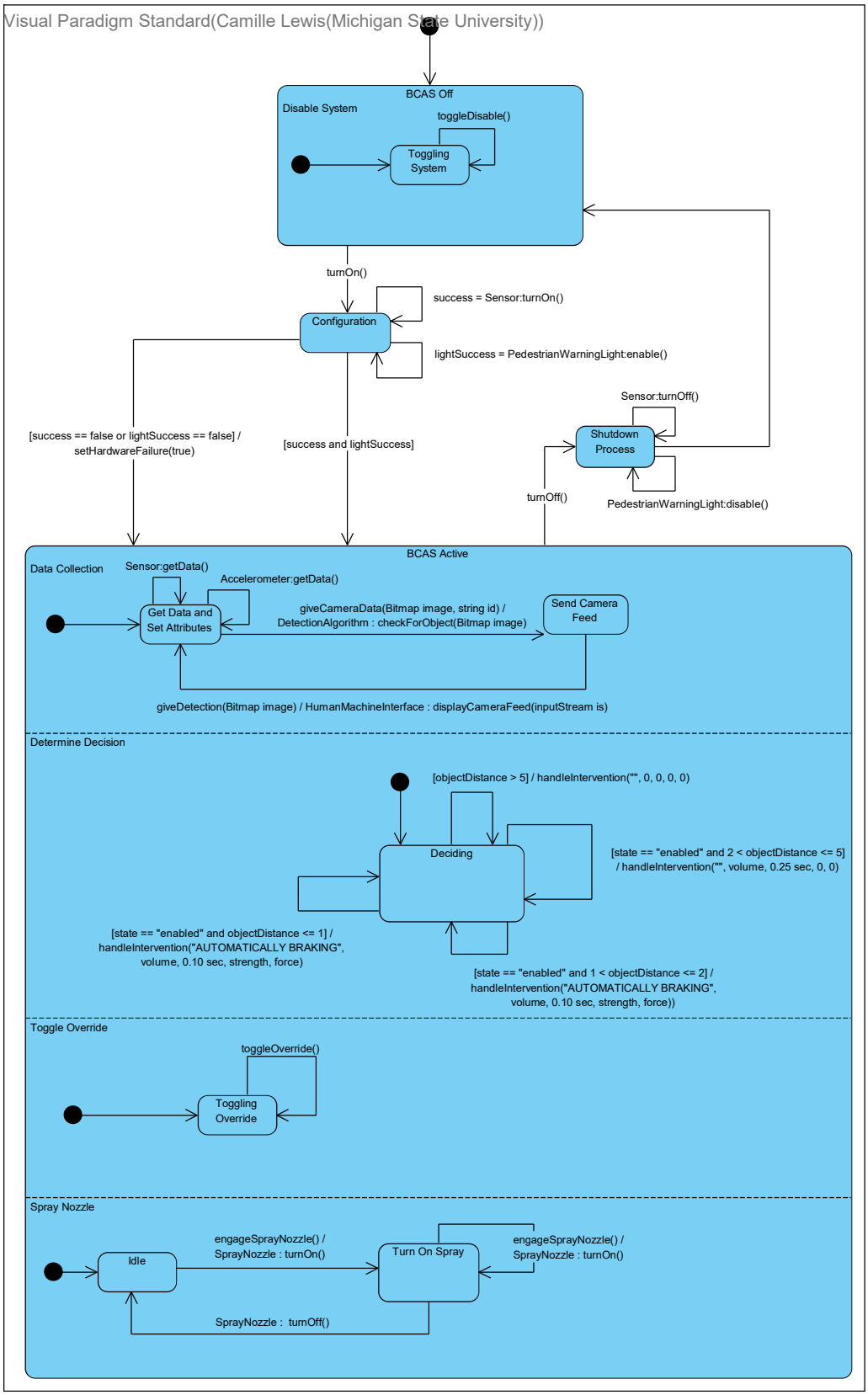

Figure 33: State diagram for the *Controller*

5 Prototype

As a proof of concept for the BCAS to be developed, we constructed a prototype framework demonstrating the features and functionalities of the BCAS system. This section introduces and demonstrates the prototype.

5.1 How to run prototype

The prototype of the BCAS can be accessed via the weblink `http://webdev.cse.msu.edu/~heinbren/backup/`. The prototype is written in PHP, Javascript, and HTML. The prototype is designed to run on the Google Chrome or the Firefox browser. The user is expected to read the instructions presented in the landing page, and proceed to the interactive prototype by using the *Go to Prototype* button.

The BCAS prototype is comprised of four main components: the road, the obstacles, the gears, and the HMI. The following subsection describes each of the components in detail.

5.1.1 Road

This subsection decomposes the component of the road and describes them in detail.

The road depicted on the right-hand side of the prototype provides a top-down view of a car backing out. While the user interacts with the prototype on the left side of the screen, the corresponding result of the action is displayed here.

The road is made up of two parts:

- **A car:** The vehicle backing out
- **A pedestrian:** The object the car is trying to avoid (must be manually added to the road using the Add Obstacle button)

5.1.2 Obstacles

This subsection decomposes the component of the Obstacles and describes them in detail.

Add Obstacles: Adds a pedestrian to the end of the road. The pedestrian can be added at any time during the backup operation. The later the obstacle is added, the more likely the vehicle is to collide with it. Even after a collision, the vehicle will apply the maximum braking force to mitigate damages in the prototype.

Add Ice: Adds ice to the cameras. As soon as the ice is added, the heated compartment automatically turns on and heats the cameras for 3 seconds, until the ice melts. It can be pressed at any time as long as there is no mud on the cameras. The heat turns on as long as the vehicle is on and is not dependent on the vehicle being in reverse. Note that in the actual system, only the main camera will be heated.

Add Mud: Adds mud to the cameras. It can only be removed by using the spray nozzle to clean the cameras. It can be pressed at any time as long as there is no ice on

the cameras. Note that in the actual system, only the main camera will have a spray nozzle.

5.1.3 Gears

This subsection decomposes the component of the gears and describes them in detail.

Reverse: Shifts the vehicle gear into reverse, turning the system on. Shifting the gear does not start the backup operation. It can be turned on and off again until the user presses the Gas button.

Gas: Starts the backup operation. It can only be clicked when the vehicle is in reverse. Once this button is activated, it cannot be turned off AND the user can not exit out of reverse for the remainder of the simulation.

5.1.4 Human Machine Interface

This subsection decomposes the component of the HMI and describes them in detail.

Disable: Disables the BCAS system. The cameras will still display an obstacle if it appears, but there will be no beeping, no warning message, no haptic feedback, and the system will not automatically brake.

Override: Does the same as disable, but it is automatically turned back on when the system goes into reverse again, and it can be toggled on/off while the vehicle is actively backing out.

Clean Camera: Activates the spray nozzle. The clean camera button must be held down continuously for a total of 2 seconds in order to completely clean the cameras. It can only be pressed while the system is in reverse.

Warning: Displays a warning to the driver that the vehicle is automatically braking, along with if it's reducing the speed by 25% or applying the max braking force.

Cameras: There are 4 cameras in the system:

$$\begin{bmatrix} \textbf{Left} & \textbf{Main} & \textbf{Right} \\ \Leftarrow & \textbf{Wide} & \Rightarrow \end{bmatrix}$$

If Add Obstacle has been clicked, then a pedestrian with a green box around them will be displayed when the vehicle is in automatic braking range. Add Ice makes camera lenses icy, requiring heat to clear them. Add Mud makes camera lenses muddy, requiring the spray nozzle to clear them.

Haptic Feedback: Represents if the driver is experiences haptic feedback on the steering wheel. The haptic feedback message will not be in the final system, but represented by a vibration in the steering wheel.

Auditory Warning: If an obstacle is detected, then the prototype will play a beeping noise. The frequency of the beep will increase if the automatic brakes are applied. The audio of the computer must be turned on to hear the beeping.

5.2 Sample Scenarios

This section introduces six different scenarios that demonstrate the BCAS prototype. A description of the prototype scenario and the corresponding screenshots during the user interaction is shown for each scenario.

5.2.1 Scenario 1: Initialization

Figure 34 shows the scenario when the prototype is loaded, the user sees four main sections: the obstacles, the human machine interface, the gas and reverse buttons, and the road where the prototype plays out. Initially, the vehicle is on and in park. The backup cameras are off and the BCAS is disabled. There is no obstacle behind the vehicle, as well.

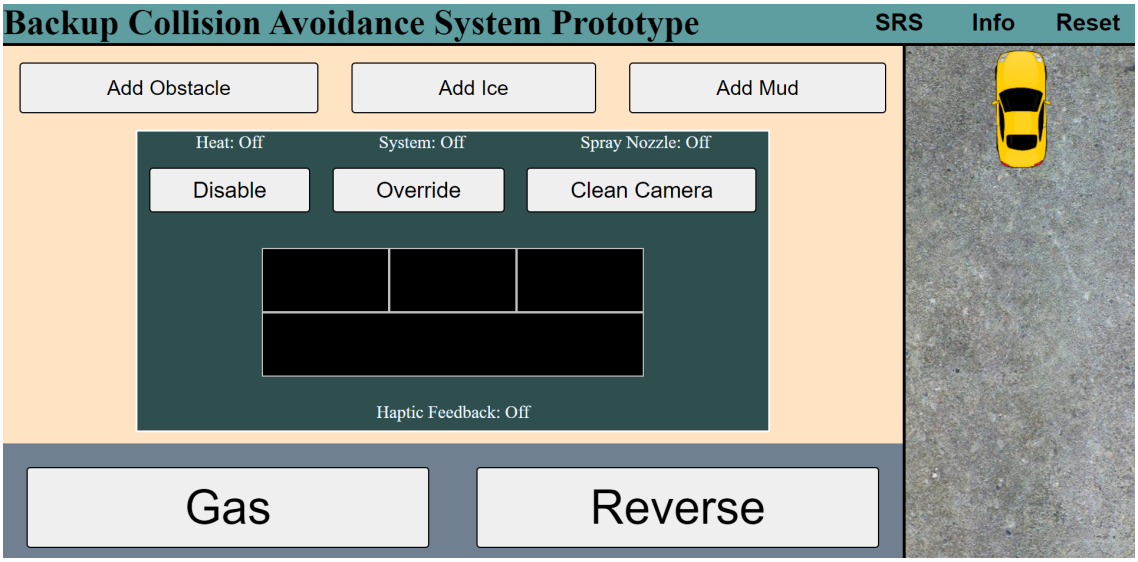

Figure 34: The prototype when first loading the page

5.2.2 Scenario 2: Add Obstacle

Figure 35 shows the scenario where the BCAS automatically brakes when detecting an imminent collision with an external variable. An obstacle is added when the user clicks the *Add Obstacle* button. Shown in the figure, the vehicle is currently backing out with a pedestrian behind the vehicle. The system is on and is actively braking. As the vehicle is not within 1 meter of the pedestrian, the current vehicle speed is reduced by 25%. The steering wheel's haptic feedback is active, as well. The backup cameras have detected the pedestrian and highlighted them in a green box. A warning beep is audible in the web demonstration prototype and is currently playing at the higher of the two frequencies in this example. Finally, a warning message appears on the HMI, as well.

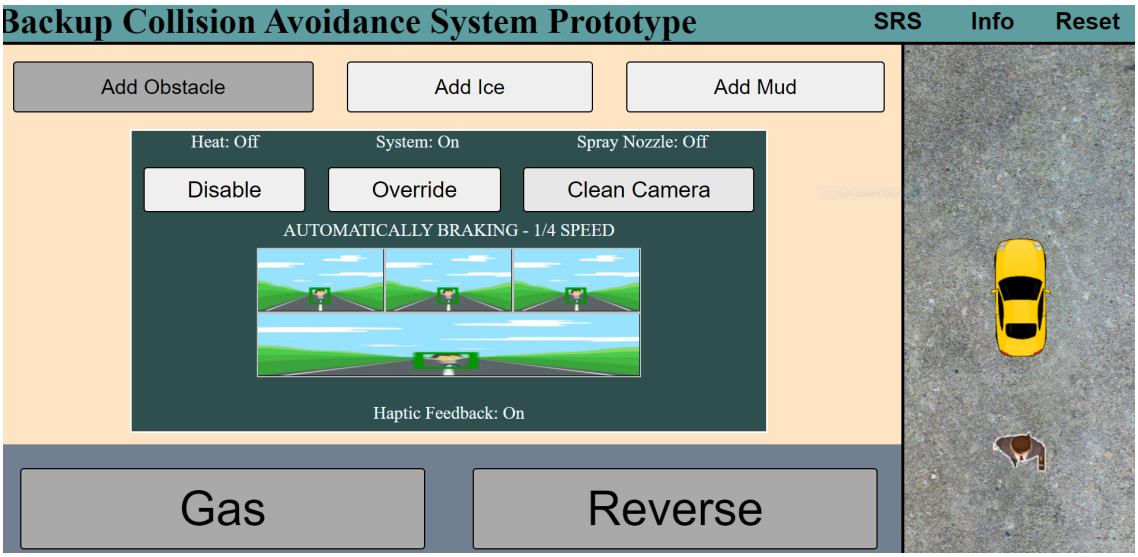

Figure 35: The prototype when the vehicle is automatically braking

5.2.3 Scenario 3: Mitigation

Figure 36 shows the scenario where the vehicle detected the pedestrian in time. The warning beep has subsided, the haptic feedback has been turned off, and the vehicle speed has been reduced to zero. The camera still displays the pedestrian, however.

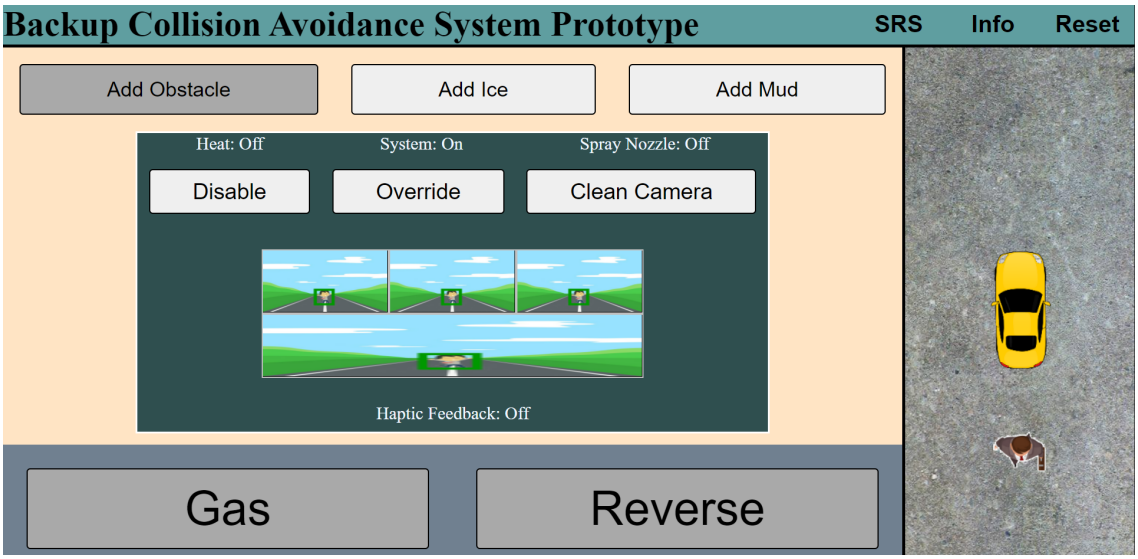

Figure 36: The prototype after a vehicle stops before colliding

5.3 Scenario 4: Collision Sensor Mitigation

Figure 37 shows the scenario where the vehicle did not detect the pedestrian in time and collides with the pedestrian. The scenario is triggered when the pedestrian is added with little time for the system to react. After colliding with the pedestrian, the collision sensor detects the accident and apply to maximum braking force to mitigate damages. The warning beep has subsided, the haptic feedback has been turned off, and the vehicle speed has been reduced to zero. The camera continues to display the pedestrian after the collision.

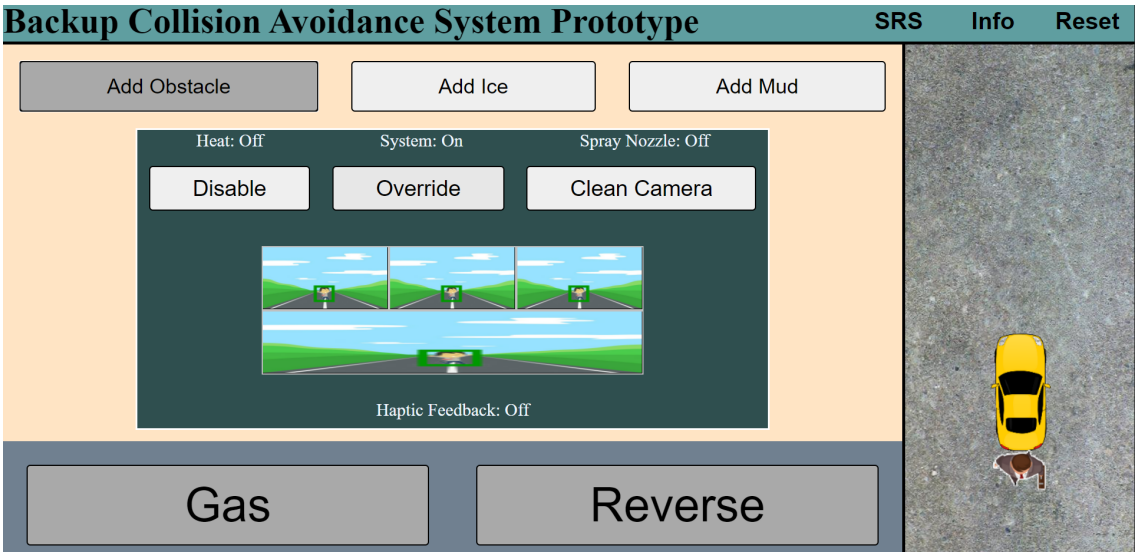

Figure 37: The prototype when the vehicle fails to stop in time and collides

5.4 Scenario 5: Heated Compartment

Figure 38 shows the scenario where the heated compartment is used to melt ice off the camera lens. By pressing the add ice button, the user can simulate ice covering the camera lens. When the cameras are covered in ice, they may not be able to detect obstacles. However, as soon as the vehicle reads a temperature of 35° or below, the heat automatically turns on. The cameras currently being heated is represented by the *Heating* message in the upper-left corner of each camera. The HMI displays the message *Heat: On* to signify that the system is currently heating the camera compartments to melt the ice, as well. After a total of 3 seconds, the heat will melt and whatever image was previously displayed on the cameras will reappear. Note that in the actual system, only the main camera will be heated.

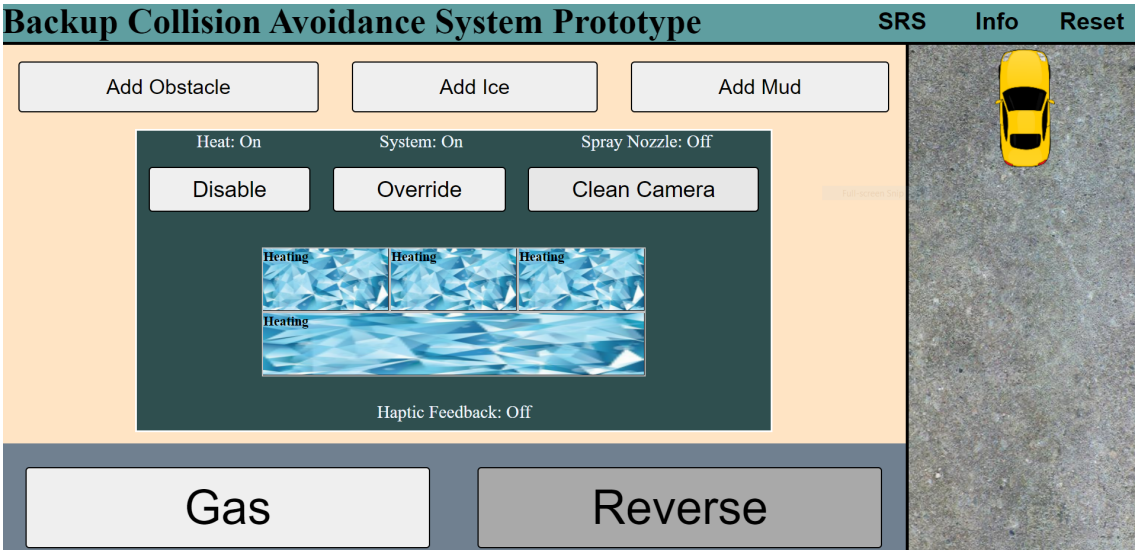

Figure 38: Example of the main backup camera used in the BCAS

5.5 Scenario 5: Mud Obscured Camera

Figure 39 shows the scenario where the camera is covered in mud. By using the *Add Mud* button, the user can simulate a dirty camera lens. When the cameras are covered in mud, they may not be able to detect obstacles. The user can remove the mud debris by holding the *Clean Camera* button continuously for a total of two seconds. While the user is holding the button to clean the camera, a *spraying* message appears in the upper-left corner of each of the four cameras, similar to the *Heating* message when there is ice on the cameras. The HMI displays the message *Spray Nozzle: On* to signify that the system is currently attempting to clear the mud, as well. Note that in the actual system, only the main camera will have a spray nozzle.

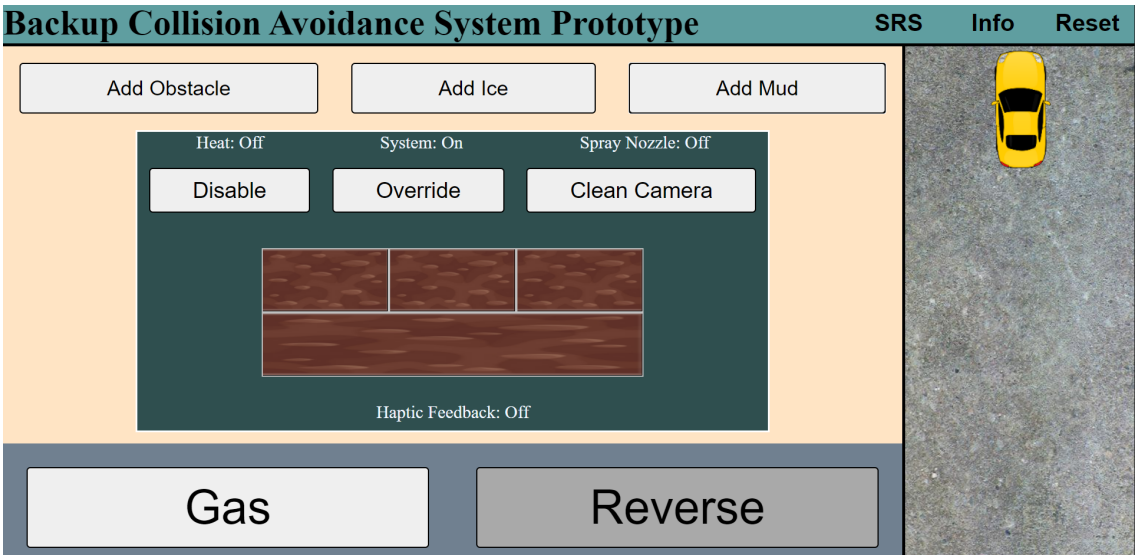

Figure 39: Example of the main backup camera used in the BCAS

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.

References

[1] Texas Department of Transportation, "Crashes by unit contributing factor." `https://ftp.txdot.gov/pub/txdot-info/trf/crash_statistics/2019/29.pdf`, 2019. Accessed: 2021-02-06.

[2] D. G. Kidd and A. Brethwaite, "Visibility of children behind 2010–2013 model year passenger vehicles using glances, mirrors, and backup cameras and parking sensors," *Accident Analysis & Prevention*, vol. 66, pp. 158–167, 2014.

[3] B.H.C. Cheng, "CSE 870 Advanced Software Engineering, in-lecture," March 2021.

[4] ISO Standard, "ISO 11898, 1993," *Road vehicles–interchange of digital information–Controller Area Network (CAN) for high-speed communication*, 1993.

[5] S. B. Smithline, "Collision avoidance system," Mar. 31 1998. US Patent 5,734,336.

[6] D. G. Studt and R. M. Taylor, "Vehicle back-up aid system," July 15 2003. US Patent 6,594,614.

[7] E. Coelingh, A. Eidehall, and M. Bengtsson, "Collision warning with full auto brake and pedestrian detection - a practical example of automatic emergency braking," in *13th International IEEE Conference on Intelligent Transportation Systems*, pp. 155–160, 2010.

[8] T. Gandhi and M. M. Trivedi, "Pedestrian protection systems: Issues, survey, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 3, pp. 413–430, 2007.

[9] A. van Lamsweerde and E. Letier, "From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering," in *Radical Innovations of Software and Systems Engineering in the Future. Lecture Notes in Computer Science (RISSEF 2002)*, vol. 2941, Springer-Verlag, 2004.

[10] National Highway Traffic Safety Administration, "Federal Motor Vehicle Safety Standard, Rearview Mirrors; Federal Motor Vehicle Safety Standard, Low-Speed Vehicles Phase-In Reporting Requirements." https://www.govinfo.gov/content/pkg/FR-2010-12-07/pdf/2010-30353.pdf, Dec. 2010.