Cruise Control Software Development Document

Team Puppies

David Liang, Aparajita Rana, Susmitha Shailesh, Max Shi

Version 2.04

12 May, 2020

Table of Contents

Section 1: Executive Summary

We aim to develop the software for a cruise control system that would include all the main features of cruise control a consumer would typically expect, such as the capability to set the desired speed, adjust the speed already set while in operation, as well as save a set speed after disabling cruise control. It will incorporate the standard sensors and physical hardware most cruise control systems use in order to regulate the speed of the car. However, the system will also be robust enough to handle unexpected interruptions to the control equipment, given the ramifications of implementing the system into real-world scenarios in order to provide the safest and most user-friendly cruise control possible. Overall, we aim to develop a fully-operational cruise-control software with an agile design process to meet the needs of users of cruise-control in a fast, reliable manner in an exhaustive number of scenarios.

Section 2: Introduction

Today, cruise control systems have become commonplace in most vehicles. With the ability to maintain a certain speed without the need to keep the accelerator held down, cruise control has made the lives of drivers significantly easier. That being said, there are obviously pros and cons to cruise control. The obvious advantage to using cruise control is that it reduces driver fatigue when on highways or sparsely populated roads. It also avoids any potential of speeding and utilizes fuel more efficiently. In regards to drawbacks, cruise control cannot be used in certain weather conditions or winding roads and could promote distracted driving. Therefore, in developing a cruise control, we would ideally seek to solve the listed cons as much as possible. With the recent popularity of autonomous vehicles, there are many more resources regarding cruise control than there were in the 1960s when cruise control just started.

While the main purpose of cruise control is simply to maintain a desired speed, there are also many other basic functionality requirements we must account for as well. For example, the cruise switch can typically consist of multiple commands including, but not limited to, an on and off switch, a way to change the set cruise control speed, and the ability to temporarily stop and resume cruise control. We must also consider how cruise control may interact with the car in its entirety. If the driver were to brake while the cruise control was still active, it would have to suspend itself until the driver resumes cruise control, such as by engaging the system again. Other features could be setting a minimum speed requirement to be able to use cruise control, or refusing to engage cruise control when the car's sensors report it is unsafe to do so, such as low tire pressure.

We aim to develop the software required to interpret input from human interfaces and data from internal automotive sensors in order to send signals to the car's powertrain and braking system to modulate speed. The physical interfaces outside of the software are outside the scope of this project, and this project will act as a black box for the inputs and outputs in the rest of the vehicle. In addition to the core systems of engaging and disengaging cruise control as well as maintaining speed, the system will be robust enough to handle unexpected situations, such as abrupt disconnection of power, damage to certain control systems, or rapid failure of systems, such as in a collision. Overall, we aim for zero failures, as this is a mission-critical piece of software with lives at stake, as well as steady and predictable operation of the cruise control when presented with multiple situations.

In order to succeed in this project, we will have to first have a thorough understanding of the mechanisms of embedded systems and cruise control in particular. We have decided what the main requirements and goals of our product will be, both for consumers and makers of the product. We have thought of use cases for our software, including what users expect from a cruise control system and how the system will be applied in practical cases. We have done research into similar systems already on the market and their strengths and weaknesses, and chosen to proceed with a call-return architecture for our product. When designing our product, we kept in mind the requirements that we had set and the use cases that the product must satisfy. Finally, we tested our product thoroughly and cycled through the process until we created a viable final design. Overall, we utilized this sort of "agile" design process when approaching this project, in order to keep costs down, be flexible to consumer demands, and ensure smooth progress in the project.

Section 3: Project Requirements

Functional Requirements

1. The cruise control software shall have four different states.
    a. The first state shall be when the software and system are turned off and are not ready to function, such as when the engine is off.
    b. The second state shall be when the software and system are on when the engine starts; the cruise control shall be ready to be set at a specific speed.
    c. The third state shall be when the software is activated; the cruise control shall maintain the set speed.
    d. The fourth state shall be when the software is suspended, such as well the throttle is changed when the system is on.
2. The Cruise Control system should check that basic functionalities of the car, through testing connections with sensors and input devices as well as the EMS, is operational for the safety of the passengers and driver on startup.
    a. The tire pressure sensors must report adequate tire pressure for driving. (This can be updated by the manufacturer, as different cars require different tire pressures)
    b. The EMS must not report any critical errors from its own system checks when the car starts.
    c. The Cruise Control system should check that all levers and buttons to operate the cruise control system are connected properly.
3. The cruise control shall only activate if a minimum speed requirement of twenty miles per hour is met.

4. The cruise control system will remain powered on through the car's battery for one minute after the engine has been shut off.
    a. The system will use this time to log deactivation of the system. The logging capabilities are detailed in the "Output" section of these requirements.
    b. The cruise control system will power off one minute after the engine has been shut off.
5. In case of unexpected events such as abrupt power shutdown, car engine issues, or running out of gas, the cruise control system shall display a warning to the driver that there is an issue with the system and it is about to turn itself off. Then, it will turn itself off.

Input

1. The cruise control system shall receive input from the driver through articles of the car's hardware, such as buttons or levers, that it should be turned on.
2. The cruise control shall receive the current speed as an input upon activation through the EMS, every time it activates.
3. The cruise control system shall increase or decrease the desired cruise speed by receiving input from the driver through two buttons that change the speed by 1 mph in the desired direction.
    a. The only increments in this cruise control system will be 1 mph up/down at a time.

4. The cruise control system shall receive information from sensors that the brake has been applied by the driver. If the driver brakes, the software shall move back into the second state (ready for activation).
6. If the driver deactivates the system, the software shall also move back into the second state.

Output

1. When the driver activates the system, the system shall send an activation request to the EMS.
2. There shall be visual feedback to inform the driver that cruise control has effectively been turned on.
3. In the activated state, the cruise control shall communicate with the throttle through the EMS to speed up or slow down to maintain the set speed. (Design goal: within 2mph)
4. The cruise control shall display that it has been turned off and why it has been turned off (driver input, unexpected event) to the driver.
5. The cruise control will log and timestamp cruise control activation, the set speed, changes in speed, and the deactivation of the system.
6. When looking to deactivate, the cruise control should provide a deactivation request to the throttle EMS.

System Requirements

1. The cruise control system shall incorporate 1 GB of memory storage for logging capabilities.

2. The cruise control software requires the necessary connections to sensors observing the road and driving conditions, the engine management system via the throttle, and human interfaces (buttons, levers, etc) to function properly.
3. Cruise control hardware shall be able to receive power both from battery and alternator.
4. The cruise control hardware shall have a system clock for logging purposes.
5. An authorized technician shall be able to download the system logs from the software.
6. The system will activate itself within half a second after the activation signal has been sent by the user.
7. The cruise control software will follow the communication protocol specified by the manufacturer, industry, and government in order to keep things safe and secure.

Performance Requirements

1. The cruise control software shall turn on and be ready to set a speed within 2 seconds of receiving the input from the driver.
2. The cruise control shall begin maintaining the speed it received as an input at startup within 1 second of activation.
3. The cruise control shall acquire and begin maintaining the new desired speed within 1 second of receiving the driver's input for a speed change.
4. The cruise control system shall log information about the state of the system within 1 second after the shutdown signal or after deciding that shutdown is necessary due to unexpected events.

Reliability Requirements

1. The software shall meet 99.999% reliability, excluding the reliability of the hardware.

Security Requirements

1. The software shall not have any access to or interfaces with wireless systems such as the internet, Bluetooth, or radio.
2. The software shall be password protected to ensure that only authorized technicians can service the software and the system.

Section 4: Use cases and UMLs

Use Cases
Use Case 1: User sets cruise control.
Actor: Driver
Goal: Set cruise control to the current speed of the car.
Preconditions: Car and cruise control is correctly turned on and activated. Additionally, the car's speed is above 25 mph.
Trigger: The driver pushes the "Set" button.

1.) The driver turns on cruise control through the "set" button.
2.) The speed sensor reads the current speed and logs it as the target cruise control speed (If there is a speed already set it overwrites it) and sends a signal back to the cruise control software.
3.) The actuator reads the logged speed and manipulates the throttle body to maintain the set speed and then returns a signal back to the cruise control software.
4.) The cruise control system provides visual feedback that cruise control has been successfully set.

5.) If the speed sensor reads a speed faster than the set speed, the actuator will be alerted and close the throttle body until the set speed is reached. If the car is too slow, then the throttle body will open and speed up the car.

6.) The speed sensor constantly reports the current speed and adjusts accordingly.

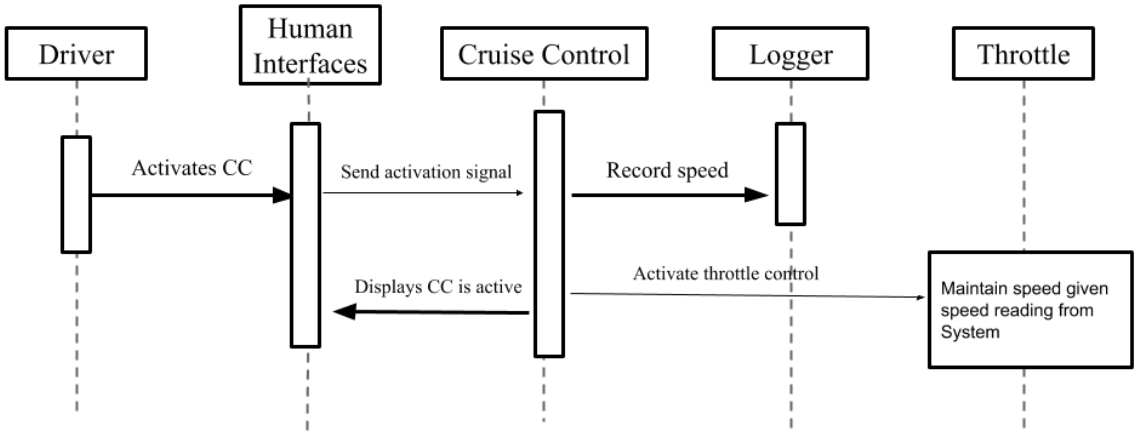
Exception Use Case

1.) The speed sensors do not send back a signal, cruise control will not activate and provide visual feedback that cruise control can not be set.

2.) The actuator does not send back a signal, cruise control will not activate and provide visual feedback that cruise control can not be set.

3.) Car has not met minimum speed and will provide visual feedback that the car must be driving at 25 mph or faster to activate.

4.) If the speed can't be logged, it will provide visual feedback that something is wrong and cruise control will not turn on



Use Case 2: User wishes to increase or decrease the set speed.
Actor: Driver

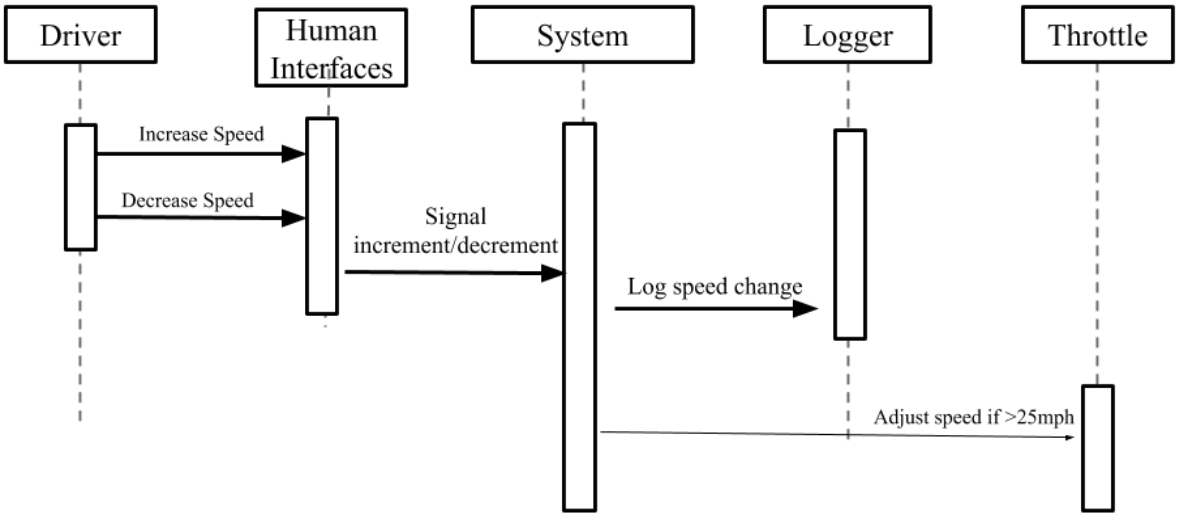
Goal: Increase the set cruise control speed
Preconditions: Cruise control is on and a speed has been set and the set speed must be above 25 always be above 25 mph.
Trigger: The driver selects the “+” or “-” buttons.

1.) The user will select “+” or “-” to increase or decrease the speed respectively.
2.) If the user taps “+”, it will increase the current logged speed by 1 mph and return a signal back that it has successfully done so. Vice versa for “-”.
3.) The speed sensor should be constantly reporting the speed and will report the current speed to the actuator.
4.) The actuator reads the set speed and adjusts the throttle body to achieve the new set speed and sends visual feedback that the cruise control has reached the new set speed.
5.) Cruise control should respond the same for “-” except the set speed decreases by 1 mph per button push.
6.) Speed sensors continuously return current speed and adjusts accordingly.

<u>Exception Use Case</u>

1.) Actuator or speed sensor does not send a signal back, cruise control will remain at its current speed and immediately provide feedback that it could not increase/decrease the speed.

2.) If the driver attempts to decrease speed while the set speed is 25 mph, it will not decrease and will provide visual feedback that cruise control is already set to the minimum speed.



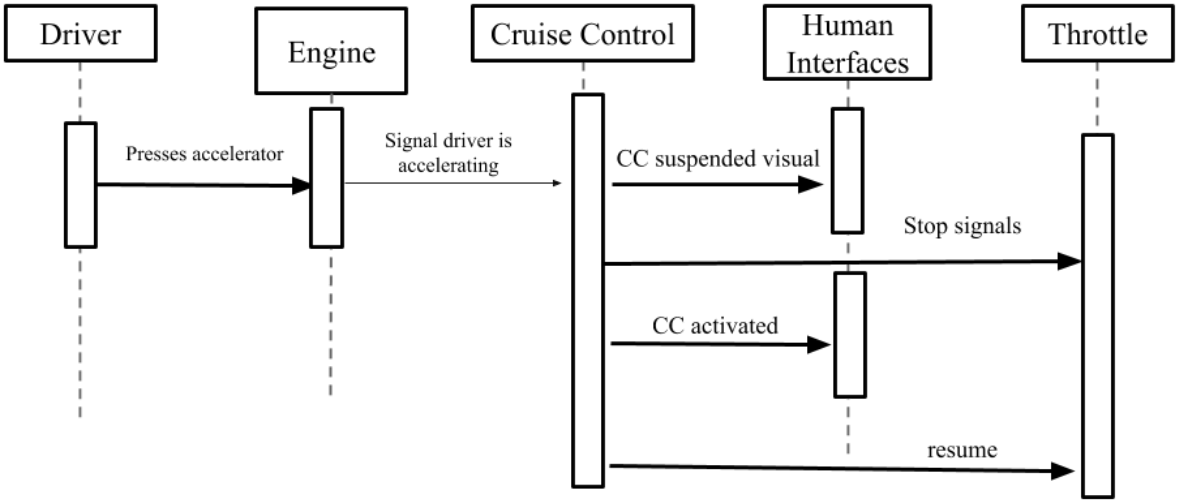
Use Case 3: User accelerates while cruise control is on
Actor: Driver
Goal: Cruise control will suspend itself and save the previous set speed
Preconditions: Cruise control has been turned on correctly and a speed has been set.
Trigger: User accelerates

1) User pushes the accelerator while cruise control is active
2) Cruise control stops sending signals to the throttle to allow the driver to accelerate freely
3) Cruise control provides visual feedback that it has been suspended
4) Cruise control remains on however, and the previous set speed is still logged.
5) If the accelerator is released, cruise control once again resumes at the previous logged speed and the throttle adjusts until the previous set speed is met.
6) If cruise control is turned off, or the brake is pressed at any time, refer to use case 4 and 5 respectively.

Exception Use Case

1.) The actuator does not respond and cruise control is turned off

2.) If speed can not be logged, it will provide visual feedback that the speed could not be logged and cruise control does not resume after the accelerator is released.



Use Case 4: User brakes while cruise control is set
Actor: Driver
Goal: Cruise control is temporarily suspended
Preconditions: Cruise control is on and a speed has been set
Trigger: Driver brakes

1.) The user brakes while cruise control is active

2.) Cruise control recognizes that the car is braking and sends a signal to the actuator.

3.) The system stops maintaining speed by letting go of the throttle, leaving the car to coast.

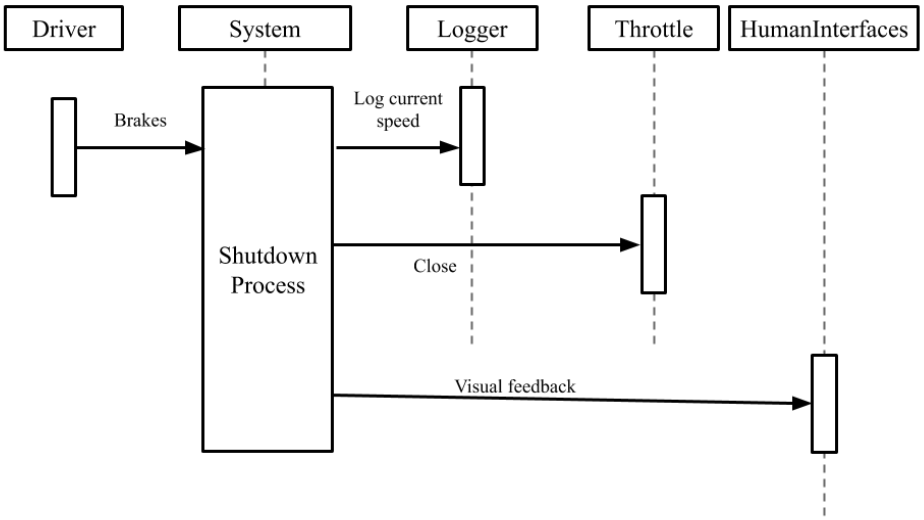
4.) The speed is logged in the memory.

5.) Cruise control sends visual feedback that it has been deactivated.

6.) If the driver selects “resume” cruise control turns back on at the previously set speed.

Exception Use Case

1.) None, if the driver brakes, cruise control should always deactivate itself

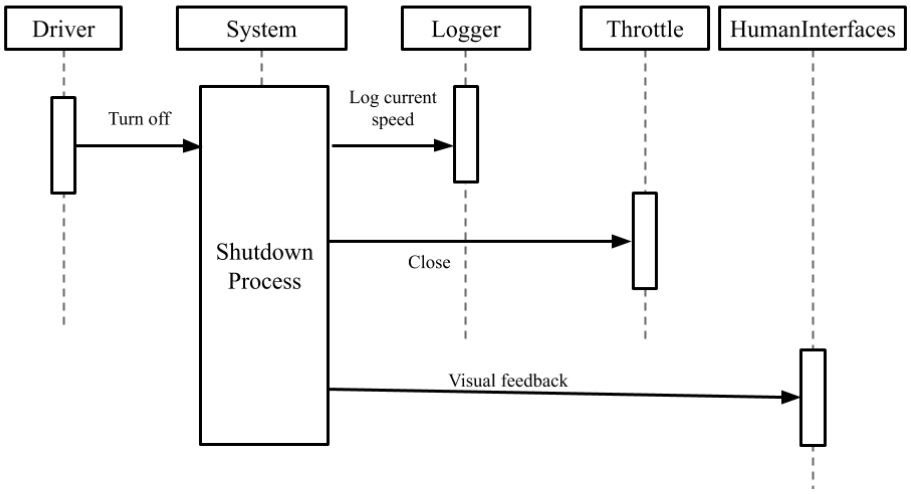


Use Case 5: User turns off cruise control
Actor: Driver
Goal: Cruise control turns off
Preconditions: Cruise control is on
Trigger: Driver turns off cruise control

1) The user turns off cruise control.
2) Cruise control logs data about the session before shutoff in the system memory.
3) The system stops maintaining speed by letting go of the throttle, leaving the car to coast.
4) Cruise control provides visual feedback that it has been turned off.
5) The car returns to the ready state to activate cruise control on the activation command.

Exception Use Case

1.) None, cruise control should always turn off no matter what happens

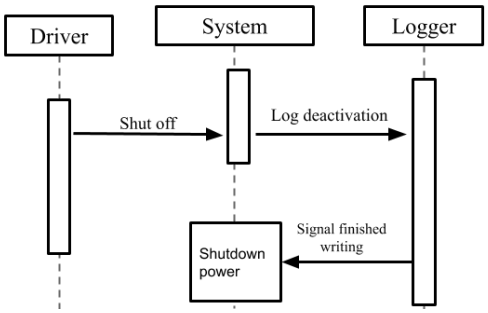


Use Case 6: User turns off car
Actor: Driver
Goal: Turn off cruise control
Preconditions: Cruise control is on
Trigger: The driver pushing the "On/off" button

1) The user shuts off the engine of the car.
2) Cruise control receives power from the car battery temporarily.
3) Cruise control logs data about the session before the engine shuts off in the system memory.
4) Cruise control completely disconnects from power after 1 minute.

<u>Exception Use Case</u>

1.) None



Use Case 7: Technician Access
Actor: Admin
Goal: Allow admin access to the software
Precondition: Admin must enter a password
Trigger: Admin must physically connects to car

1.) Admin connects to cruise control via usb cable

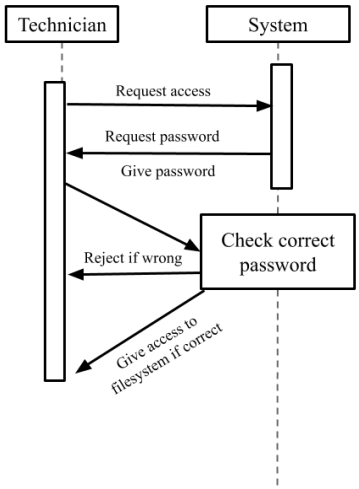
2.) Admin must enter a password to gain full access

3.) Admin can download and view data previously logged in the memory.

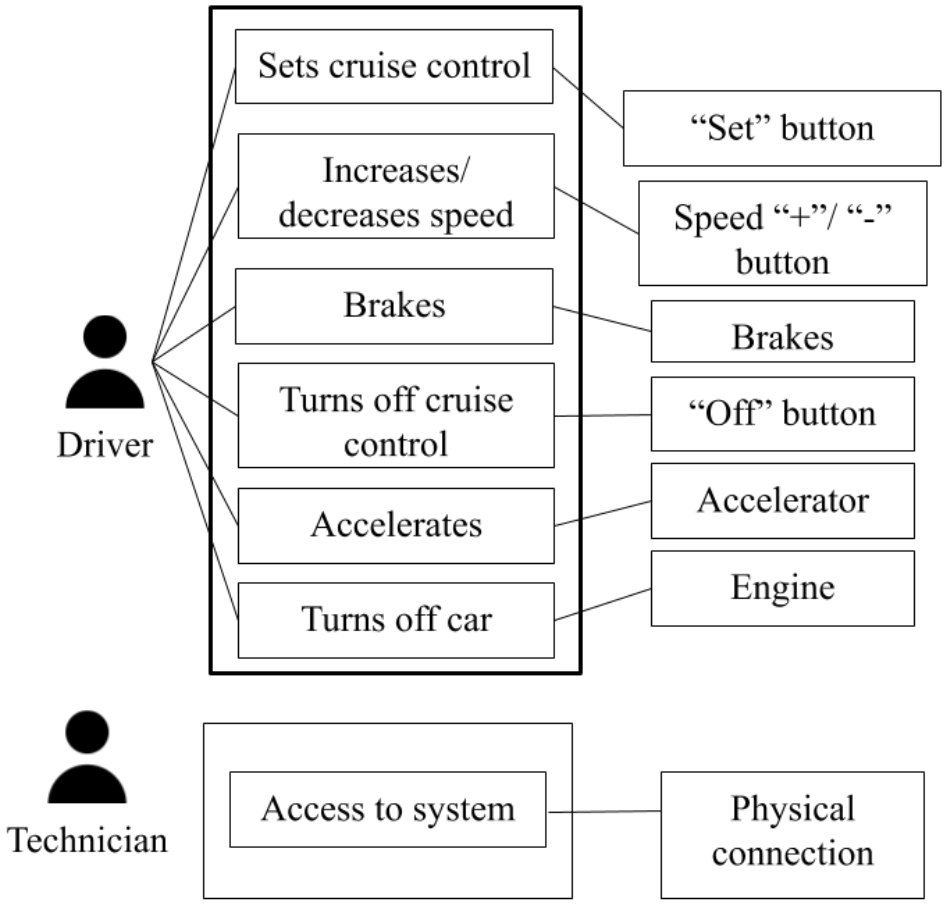
4.) Admin can update the software

Exception Use Case

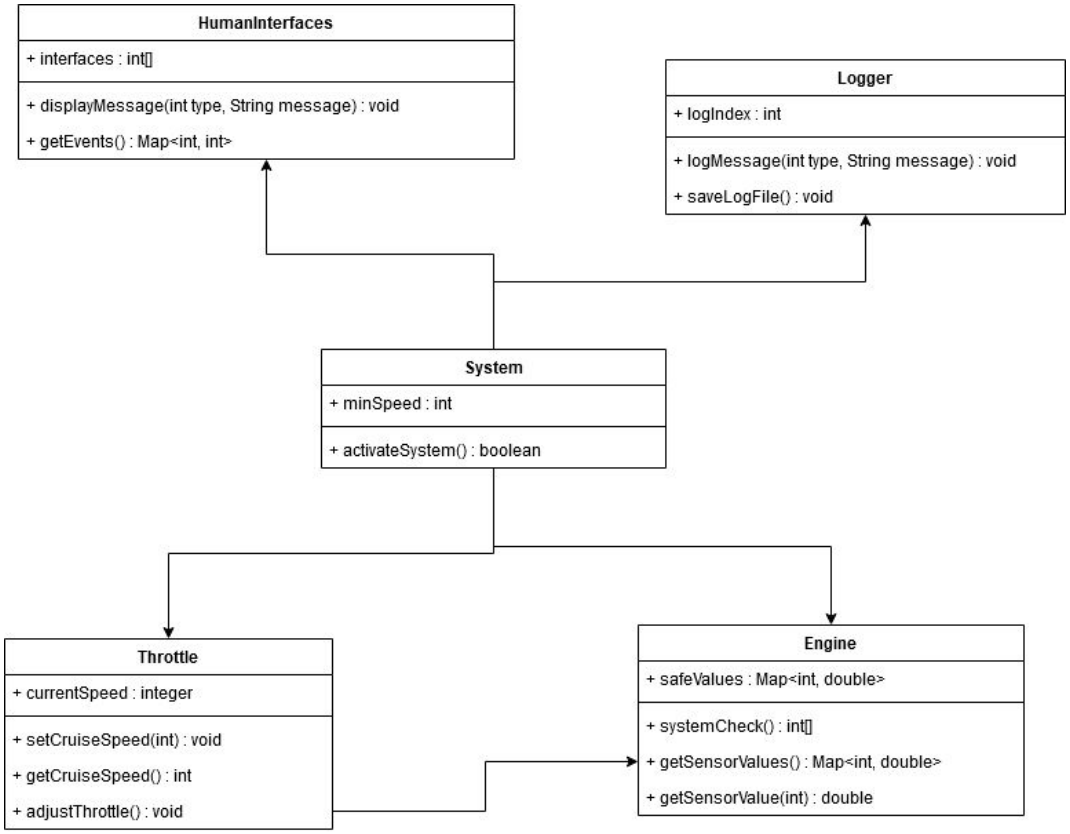
1.) 5 unsuccessful attempts to login will block the device connected and lock the system for 20 minutes.



Use Case Diagram



Class-Based UML + Index Cards



System	
Responsibilities	**Collaborators**
Gets results of system engine check and interprets them to allow activation of cruise control	Engine
Activates/deactivates cruise control on user input from HumanInterface and when Engine	HumanInterface
Activate throttle control on interface input	HumanInterface, Throttle
Monitor engine events for dangerous events to halt cruise control	Engine, Throttle
Display reasons for activation and deactivation to driver's display	HumanInterface

Log all actions	Logger

Logger	
Responsibilities	**Collaborators**
Logs output passed from other classes	All other classes
Writes output to system memory	
Timestamps log output with the system clock	

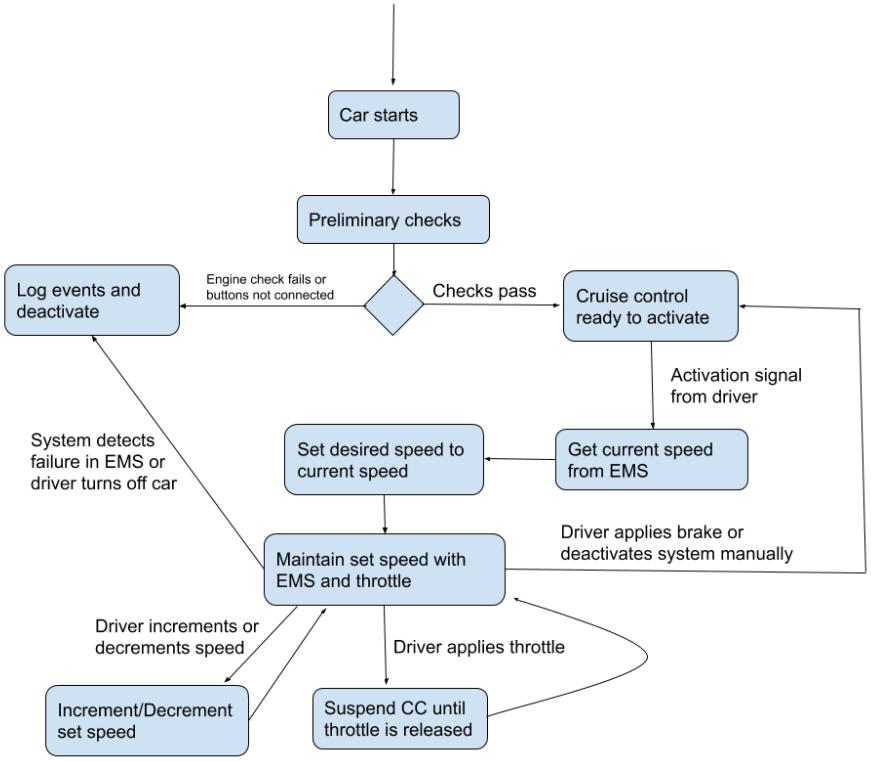
Engine	
Responsibilities	**Collaborators**
Interfaces with engine sensors to interpret safe and dangerous values.	
Raise an event to the System in case of failure.	System
Conduct overall system checks.	
Be able to get EMS sensor values when requested by other classes	System, Throttle

Throttle	
Responsibilities	**Collaborators**
Communicates with Engine sensors to modulate throttle and control speed.	Engine
Receives activation/deactivation signals from System.	System

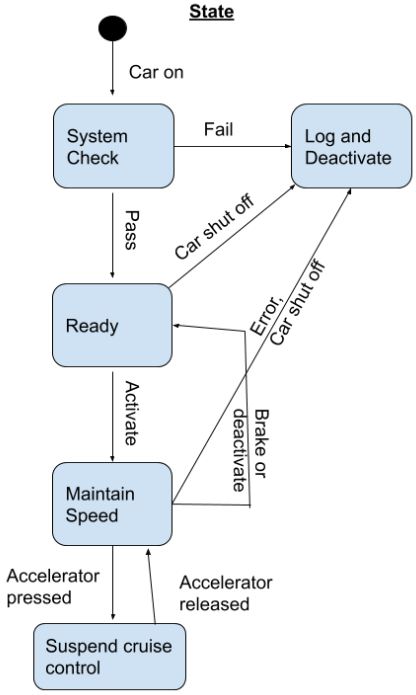
HumanInterface

Responsibilities	**Collaborators**
Monitor input from human interfaces (levers, buttons, throttle, brakes)	
Display messages to onboard car displays from System.	System
Send any inputs from driver to the System to handle.	System

Activity Diagram



State Diagram



Section 5: Software Architecture

Architectural Choice

For the software architecture styles for this project, we considered six different options, and weighed the benefits and drawbacks of each when applied to this project.

Data Centered Architecture

Data Centered Architecture utilizes a data store at the center of multiple connected clients, to distribute data between a central hub and users. This is great for large projects with a lot of users, to connect a lot of people together at the same time, and to serve multiple clients at once. However, this architecture is not relevant to this project, as the cruise control software only serves one client, the driver, and does not have a significant amount of data to pass to clients to utilize the benefits of a centralized data store.

Data Flow Architecture

Much like Data Centered Architecture, Data Flow Architecture is great at manipulating large amounts of data and sending it through many filters to arrive at a desired result at the end of the flow. While this architecture may be great for things like search engines or social networks, where a lot of data must be manipulated, there is not enough data from a car's sensors to warrant using this type of data architecture. Furthermore, this architecture does not have a two-way data flow path, which might be necessary in order for user input to make changes to the behavior of the cruise control system.

Call Return Architecture

Call Return Architecture utilizes a main program and subprograms/subclasses in order to create a two-way data path between the main program and other components of the software. This is good for creating programs that are easy to scale and modify, and the subprograms provide an encapsulation that could apply well to the different components that make up cruise control software, as they need to be able to communicate with each other in order to execute the correct behavior. The downside to this architecture would be that the main program would need to handle all the communication between subprograms, which could make the program slow and difficult to meet performance requirements.

Object-Oriented Architecture

Object-Oriented Architecture is based on the responsibilities and actions a software can do. It is most popularly seen as seeing the system is made up of a collection of various objects. It is helpful in highlighting the actual objects within the software, is reusable through

polymorphism, and is able to manage errors through execution. In terms of disadvantages, it is difficult to constrain by time and budget as well as difficult to reuse on a larger scale.

Layered Architecture

The way layered architecture works is through various layers that each have a specific function. Each layer then is further composed of different correlated components that affect the function. Due to the structure of this approach, different teams can easily be designated different layers to work on. This approach is good for large teams focused on User Interfaces; however, the main disadvantage is that if there are too many layers things become inefficient and backed up. There is also not much room for scalability and correlated changes that a project like ours may need.

Model View Controller Architecture

Model View Controller Architecture focuses on the relationship between three aspects of the project: the model, the view, and the architecture. The model is the data and logic that a given user works with. The view refers to the UI that a given user interacts with. The controller regulates which view and model a given user should utilize. For example, a customer and a vendor interact with different versions of the same webpage; the customer sees the customer view and works with customer-related data in the customer model while the vendor sees the vendor view and works with vendor-related data in the vendor view. Model View Controller Architecture is used mainly for web design projects and would not be suitable for our cruise control software.

For our cruise control software, we choose to implement the **call-return architecture**, for its ability to communicate and handle data through the main program between the different elements in the car. Other highly considered options were the layered architecture and the object oriented architecture, however, we decided that the user interface and the layers would not be complex enough to warrant a layered architecture approach, and object oriented design would make data transmission and handling very complicated to implement. Thus, even though Call Return architecture could present problems in performance when communicating data between components, we believe that its benefits are most suited to this project.

Details of our Call Return Architecture

Based on the architectural style for the Call Return Architecture, our system needs to be made up of a main program and various controller subprograms and application subprograms. Specific to the cruise control software we are building, the main program would be our System

itself consisting of the minimum speed and activation. Our human interfaces and logger would be two seperate controller subprograms as they are mostly independent of each other. Lastly, throttle and engine would make up one other controller subprogram. The throttle and engine controller subprogram would further consist of application subprograms that include things such as setting the cruise control speed, adjusting the throttle, checking the system, and getting sensor data. Throttle and Engine have connectors amongst them as many of their functions are intertwined resulting in them needing the ability to communicate, coordinate, and cooperate. The constraints would be through seeing how each block of the architecture is built; the main program is the root of the whole system with the rest of the subprograms and application subprograms being reliant on the success of the main.

Control Architecture

In Call Return Architecture, the control is centralized. This means that one component is in charge of managing the execution of other components. This architecture has a control hierarchy when one main program invokes other program components, which in turn invoke other components, and so on. The hierarchy is top-down; control starts at the top of the hierarchy with the main program and is distributed to lower levels of the tree via calls. Each subroutine of the tree can choose to either pass control down to the next level or return it up to its parent with a call return. At the end of the software running, control is returned to the point where the routine was called. The control topology looks like a tree which starts with the main program as the root and with each routine having a parent and possibly subroutines. Components operate synchronously in this architecture.

Data Architecture

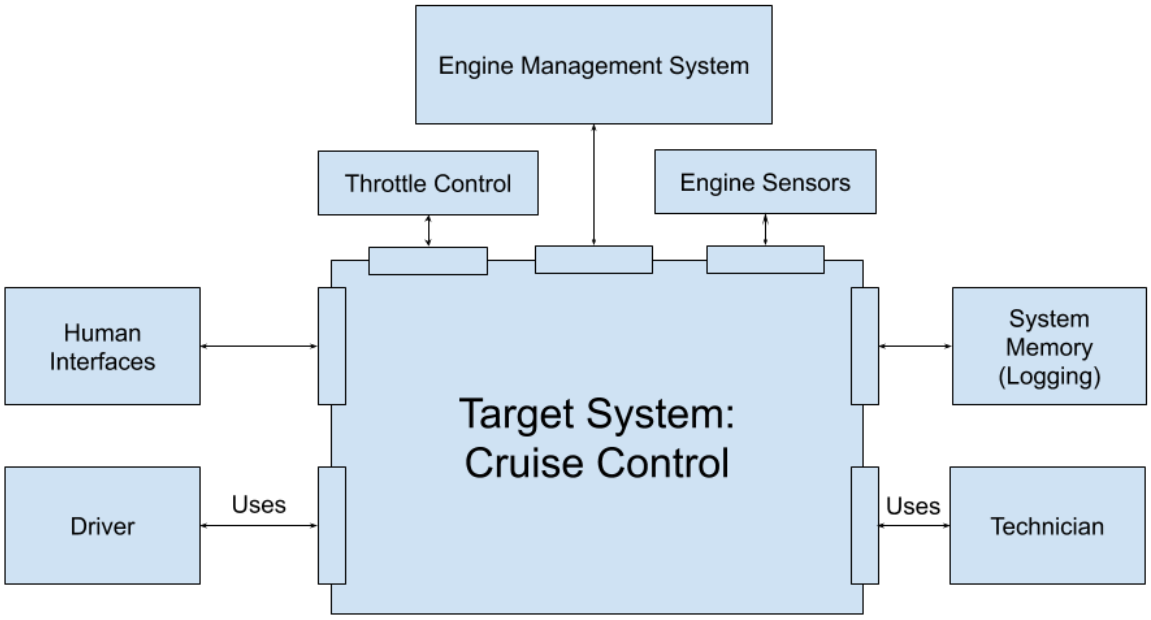
In our Cruise Control Software, data is communicated between various components with the use of sensors and interfaces to address the current status of the various parts. By utilizing these functions, we are able to exchange and receive different information. In general, the flow of data is continuous as it needs to be constantly updating in order to keep the driver safe. It is important to constantly log and record the speed so that we know if the system is working as expected. Specific objects are passed “sporadically” in terms of the driver entering a desired cruise control speed and the desire to turn off the system. We transfer data through function calls that cause other functions to come into action. Data components do exist for organization and efficiency of our program. They give us our basic structure. Functional components are often directly related to data components because the result of functional components are the data amounts. Data components regarding built in system safety are passive and can not be changed. Other components based on the current speed and other changing factors are active. Data is more

changeable while our control is our checker in a sense to protect and ensure our program is running in the desired means.

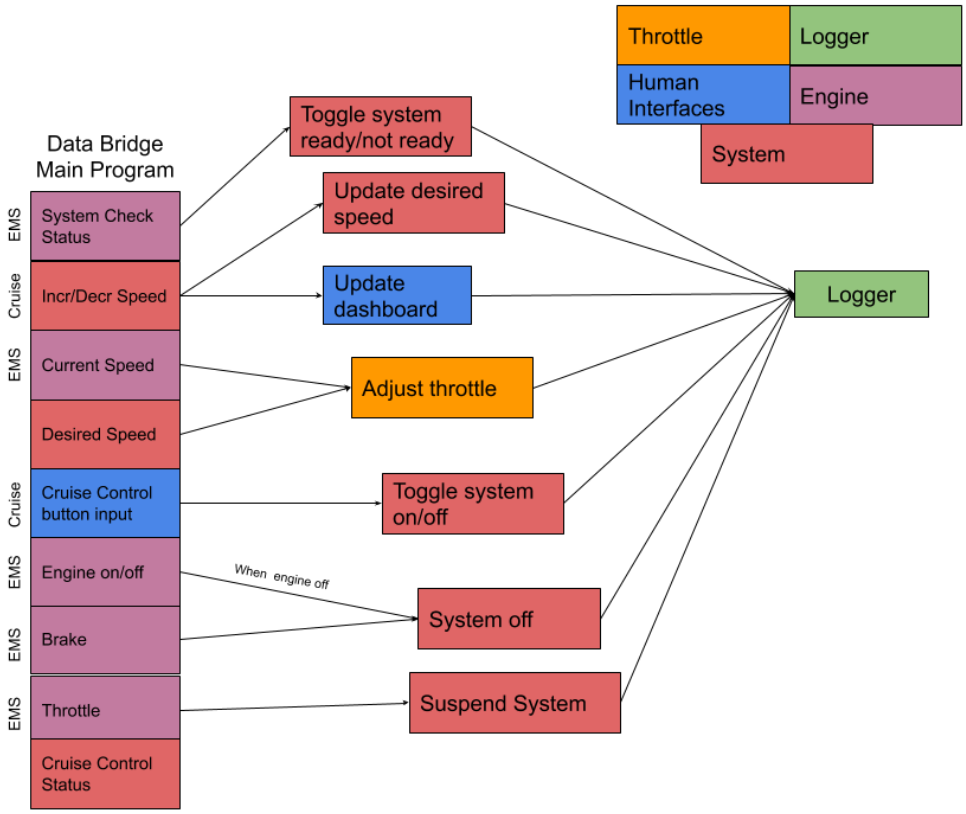
Summary of Issues in Software Architecture

The main issue that may arise is regarding the synchronous manner of the Call Return Architecture. Because the components do not function simultaneously and need to be called after the bridge updates, our software may be slowed down by this architecture, especially if all data is passed through the main program. However, it is imperative for Cruise Control software to perform quickly in order to regulate fast speeds and to ensure safety. We will have to ensure, while coding the program, to maximize efficiency and function calls and minimize delays that may occur. Especially in mission critical software like this one, we will also have to pay close attention to the methods in the main program to make sure calls happen in the right order and ensure safe and predictable behavior in edge cases.

Software Architecture Context Diagram



Software Component Architecture Diagram



Section 6: Code

code_system.py

```
import tkinter as tk
import cruisecontrol as cc
from PIL import ImageTk, Image

#data bridge, indexes should refer back to component architecture
dataBridge = [0,0,0,0,0,0,0,0,0]
#parent window
window = None
#deceleration constant
deceleration = 0
#window labels
textfields = [0,0,0,0,0,0]

#callback for changing the throttle manually
def changeThrottle(event):
    global dataBridge
    event = int(event)
    dataBridge[7] = event
    cc.suspend(dataBridge)

#callback for toggling CC system
def toggleCruise():
    print("cruise toggle ")
    global dataBridge
    dataBridge[4] = 1
    cc.toggleOnOff(dataBridge)

#callback for turning engine on correctly
def toggleEngine():
    global dataBridge
    #print("engine toggle "+ str(dataBridge[5]))\
    dataBridge[0] = 0
    if dataBridge[5] == 0:
        dataBridge[5] = 1
```

```
        cc.toggleReady(dataBridge)
    else:
        dataBridge[5] = 0
        cc.shutoff(dataBridge)

#callback for turning engine on with failure
def toggleBadEngine():
    global dataBridge
    if dataBridge[5] == 0:
        dataBridge[5] = 1
        dataBridge[0] = -1
        cc.toggleReady(dataBridge)

#callback for changing brake value
def changeBrake(event):
    global dataBridge
    event = int(event)
    dataBridge[6] = event
    cc.shutoff(dataBridge)

#callback for changing deceleration of car
def changeDecel(event):
    global deceleration
    event = int(event)
    deceleration = event

#callback for incrementing speed
def incrementSpeed():
    global dataBridge
    dataBridge[1] = 1
    cc.updateSpeed(dataBridge)

#callback for decrementing speed
def decrementSpeed():
    global dataBridge
    dataBridge[1] = -1
    cc.updateSpeed(dataBridge)
```

```
#main loop function for simulating the movement of a car
def simulateCar():
    global window, deceleration, dataBridge, textfields
    #simulate acceleration/deceleration
    if(dataBridge[5] == 1):
        cc.adjustThrottle(dataBridge)
        dataBridge[2] += dataBridge[7]*0.05 - dataBridge[6] * 0.05 + deceleration
* 0.1
        if(dataBridge[2] < 0):
            dataBridge[2] = 0

    #update text fields
    textfields[0].configure(text = "Current Throttle:
{thr:0.2f}".format(thr=dataBridge[7]))
    textfields[1].configure(text="Current Brake: " + str(dataBridge[6]))
    textfields[2].configure(text="Desired Speed: {speed:0.2f}".format(speed =
dataBridge[3]))
    textfields[3].configure(text="Current Speed: {speed:0.2f}".format(speed =
dataBridge[2]))
    textfields[4].configure(text="Engine Status:\n" + ("ON" if dataBridge[5] == 1
and dataBridge[0] == 0 else ("ON - FAIL" if dataBridge[5] == 1 else "OFF")))
    textfields[5].configure(text="Cruise Control Status:\n" + ("READY" if
dataBridge[8]==1 else ("ACTIVATED" if dataBridge[8] == 2 else ("SUSPENDED" if
dataBridge[8] == 3 else ("FAILURE" if dataBridge[8] == -1 else "NOT READY")))))
    window.after(500, simulateCar)

#callback for failing car
def raiseEmergency():
    global dataBridge
    dataBridge[0] = -1
    cc.toggleReady(dataBridge)

#initialize gui
def initGUI():
    global dataBridge
    global window
```

```
    global textfields

    #initialize main window
    window = tk.Tk()
    window.title("Team Puppies Cruise Control")
    window.geometry("800x900")
    window.configure(background='brown')

    path = "teampuppies.png"
    img = ImageTk.PhotoImage(Image.open(path))


    title = tk.Label(window, image = img)

    #create all labels and buttons
    throttle_slider = tk.Scale(
        window,
        from_=100,
        to=0,
        command=lambda x: changeThrottle(x)
    )
    brake_slider = tk.Scale(
        window,
        from_=100,
        to=0,
        command= lambda x: changeBrake(x)
    )
    decel_slider = tk.Scale(
        window,
        from_=5,
        to=-40,
        command=lambda x: changeDecel(x)
    )
    decel_slider.set(-5)
    engine_start = tk.Button(
        window,
        text="Start/Stop Engine",
```

```
        command = lambda: toggleEngine()
    )
    cc_start = tk.Button(
        window,
        text="Start/Stop Cruise Control",
        command=lambda: toggleCruise()
    )
    currentspeed = tk.Label(
        window,
        text="Current Speed: "
    )
    currentthrottle = tk.Label(
        window,
        text="Current Throttle: "
    )
    currentbrake = tk.Label(
        window,
        text = "Current Brake: "
    )
    cc_up = tk.Button(
        window,
        text="Speed+",
        command = lambda: incrementSpeed()
    )
    cc_down = tk.Button(
        window,
        text="Speed-",
        command = lambda: decrementSpeed()
    )
    throttle_slider_label = tk.Label(
        window,
        text="Throttle Slider\nMax:100\nMin:0"
    )
    decel_slider_label = tk.Label(
        window,
        text="Deceleration Slider\nMax:10\nMin:-40"
    )
```

```
brake_slider_label = tk.Label(
    window,
    text="Brake Slider\nMax:100\nMin:0"
)
desiredspeed = tk.Label(
    window,
    text = "Desired Speed: "
)
cc_status = tk.Label(
    window,
    text="Cruise Control Status:"
)
engine_status = tk.Label(
    window,
    text="Engine Status:"
)
bad_engine_start = tk.Button(
    window,
    text="Engine Start Fail",
    command = lambda: toggleBadEngine()
)
emergency_button = tk.Button(
    window,
    text = "Create Emergency",
    command = lambda: raiseEmergency()
)
#configure fonts
title.config(font=("Arial",25))

cc_status.config(font=("Arial", 14))
engine_status.config(font=("Arial", 14))

currentbrake.config(font=("Arial",16))
currentspeed.config(font=("Arial",16))
currentthrottle.config(font=("Arial",16))
desiredspeed.config(font=("Arial",16))
```

```
    #place all buttons
    title.place(relx = 0.5, rely = 0.2, anchor=tk.CENTER)

    cc_status.place(relx=0.5, rely = 0.4)
    engine_status.place(relx = 0.28, rely = 0.4)

    cc_start.place(relx=0.5, rely= 0.47)
    engine_start.place(relx = 0.3, rely = 0.47)
    bad_engine_start.place(relx = 0.1, rely = 0.47)
    emergency_button.place(relx = 0.1, rely = 0.43)

    decel_slider.place(relx = 0.2, rely = 0.83)
    throttle_slider.place(relx = 0.5, rely = 0.83)
    brake_slider.place(relx = 0.8, rely = 0.83)



    currentspeed.place(relx = 0.35, rely = 0.6, anchor=tk.CENTER)
    currentthrottle.place(relx = 0.35, rely = 0.65, anchor=tk.CENTER)
    currentbrake.place(relx = 0.35, rely = 0.7, anchor=tk.CENTER)
    desiredspeed.place(relx = 0.35, rely = 0.55, anchor=tk.CENTER)
    cc_up.place(relx = 0.35, rely = 0.75)
    cc_down.place(relx = 0.55, rely = 0.75)

    throttle_slider_label.place(relx = 0.38, rely= 0.83)
    decel_slider_label.place(relx = 0.03, rely=0.83)
    brake_slider_label.place(relx = 0.68, rely = 0.83)

    #initialize more globals and start main loop
    dataBridge[5] = 0
    textfields = [desiredspeed, currentspeed, currentthrottle, currentbrake,
engine_status, cc_status]
    window.after(100, simulateCar)
    window.mainloop()
```

```
if __name__ == '__main__':
    initGUI()
```

logger.py

```
#datetime to get the timestamp
import datetime
#create global logfile
logfile = None

#get file name -> create and open file
def init(filename):
 global logfile
 logfile=open(filename,"a+")

#do the actual logging
def log(message):
 global logfile
 #timestamp
 time=datetime.datetime.now().ctime()
 #combine timestamp and message and insert on line
 logfile.write("["+time+"]:\t"+message+"\n")

#close file
def shutdown():
 global logfile
 logfile.close()
```

cruisecontrol.py

```
import logger as log

# bridge[8] = system status, 0 not ready, 1 is ready, 2 is activated, 3 is
suspended
decconstant = 5
def toggleReady(bridge):
```

```
    # when engine is turned on, cruise control set to ready
    log.init("logfile.txt")
    if bridge[5] == 1 and bridge[0] == 0:
        log.log("System ready.")
        bridge[8] = 1
    elif bridge[0] != 0:
        log.log("Engine fail, system not ready.")
        bridge[8] = -1
        bridge[7] = 0

def updateSpeed(bridge):
    # +1, 0, -1, desired speed updated to reflect input
    if(bridge[1] > 0):
        bridge[3]+=1
        log.log("Speed increased to "+str(bridge[3])+".")
    elif(bridge[1] < 0 and bridge[3] > 20):
        bridge[3]-=1
        log.log("Speed decreased to "+str(bridge[3])+".")
    bridge[1] = 0

def adjustThrottle(bridge):
    # 100 = Throttle is uncapped, 0 = Throttle closed
    global decconstant
    # desired > current increase throttle, desired < current decrease throttle
    if bridge[8] == 2:
        diff = abs(bridge[3] - bridge[2])
        if bridge[2] < bridge[3]:
            bridge[7] = diff*decconstant
            decconstant+=0.5
            log.log("Throttle opened to increase current speed to match desired
speed")
        elif bridge[2] > bridge[3]:
            decconstant-=0.5
            bridge[7] = 0
            log.log("Throttle closed to decrease current speed to match desired
speed")
        if bridge[7]<0:
```

```
            bridge[7] = 0
        elif bridge[7] > 100:
            bridge[7] = 100

def toggleOnOff(bridge):
    # 1 = cruise control button input pressed
    if(bridge[4] == 1 and bridge[2] >= 20 and bridge[8] == 1):
        bridge[8] = 2
        #set desired speed to current speed
        bridge[3] = bridge[2]
        log.log("Cruise control turned on.")
        log.log("Speed set to "+str(bridge[3])+".")
    elif(bridge[8] == 2):
        bridge[8] = 1
        bridge[4] = 0
        log.log("Cruise control turned off.")

def shutoff(bridge):
    if(bridge[5] == 0):
        #engine off
        bridge[8] = 0
        log.log("Cruise control turned off.")
        bridge[7] = 0
    elif(bridge[6] == 1):
        #brake pressed
        bridge[8] = 1
        log.log("Cruise control turned off.")
        bridge[7] = 0

def suspend(bridge):
    if(bridge[7] > 0 and bridge[8] == 2):
        #throttle > 0
        bridge[8] = 3
        log.log("Cruise control suspended.")
    elif(bridge[7] == 0 and bridge[8] == 3):
        bridge[8] = 2
        log.log("Cruise control unsuspended")
```

Section 7: Test Cases

Function Requirements

1. Test that all four states function.
    a. The system state when the engine is turned off should be NOT READY.
    b. The system state immediately after the engine is turned on should be READY.
    c. The system state when the cruise control button input is toggled on should be ACTIVATED. Desired speed should be set to the current speed.
    d. The system state when the throttle is changed while the system state is ACTIVATED should be SUSPENDED.
2. If the system state is READY, the functionality of the car must be true. (Because we are not working with any actual hardware, we cannot genuinely check if the car we are working with is functional. We have a makeshift check functionality function that always returns true.)
3. If the current speed is less than 20, system state can not be changed to ACTIVATED.
4. Test system actions after engine shutoff.
    a. After deactivation, the deactivation of the system must be documented in log.
    b. Within one minute after engine shutoff, the system state must be NOT READY.
5. When the emergency button is clicked, a warning must be displayed to the driver and the system state must be changed to NOT READY.

Input

1. When the Stop/Start Cruise Control button is pressed when the cruise control status is READY, the status should be changed to ACTIVATED.

2. When the Stop/Start Cruise Control button is pressed when the cruise control status is READY, the desired speed should equal the current speed.
3. When the Speed+ button is pressed, the desired speed must be increased by 1. When the Speed- button is pressed, the desired speed must be decreased by 1.
4. When the brake slider is increased past 0 when the system state is ACTIVATED or SUSPENDED, the system state must be changed to READY.
5. When the Stop/Start Cruise Control button is pressed when the cruise control status is ACTIVATED, the status should be changed to READY.

Output

1. When the driver activates the system, the system shall send an activation request to the EMS.
2. When the system state is changed to ACTIVATED, a message must be displayed to inform the driver.
3. When the system state is ACTIVATED, the current speed should always be within 2mph of the desired speed.
4. When the system state is changed to READY from ACTIVATED, a message must be displayed to inform the driver.
5. After the system is turned on, the log must be updated. After the speed is increased or decreased, the log must be updated. After the system is turned off, the log must be updated.
6. When looking to deactivate, the cruise control should provide a deactivation request to the throttle EMS.

System Requirements

1. Because we are not working with any actual hardware, we have no way of testing this requirement. For our purposes, we will assume that this requirement is met.
2. The interfaces in our software (buttons, sliders, etc.) must be functional and have an effect on the system.
3. Because we are not working with any actual hardware, we have no way of testing this requirement. For our purposes, we will assume that this requirement is met.
4. The log must include a timestamp for every entry.
5. The log must be able to be downloaded. In our case, it is saved as a .txt file and can be downloaded off the computer.
6. The system state must be changed to ACTIVATED within half a second after the stop/start cruise control has been pressed.
7. For our purposes, we will assume that this requirement is met.

Performance Requirements

1. The system state must be changed to ACTIVATED within 2 seconds of the Stop/Start cruise control button being pressed.
2. The desired speed must equal and remain at the current speed within 1 second of the system state being changed to ACTIVATED.
3. The current speed must equal and remain at desired speed within 1 second of the Speed+ or Speed- buttons are pressed.
4. After the system state is changed to NOT READY, an entry must be added to log recording the shutdown state.

Reliability Requirements

1. The software must pass all written test cases.

Security Requirements

1. Because we are not working with any actual hardware, we have no way of testing this requirement. For our purposes, we will assume that this requirement is met.
2. Because we are not working with any actual hardware, we have no way of testing this requirement. For our purposes, we will assume that this requirement is met.

Section 8: Issues

Raised in Review

1. Lost labels from class diagram in the software component architecture diagram.
2. Lack of consistency between context diagram and component diagram.
3. No mention of testing interface in document.
4. Class-based UML is outdated for a call-return architecture.
5. Inconsistent mentions of configuration file.
6. Introduction needs to relate more to the overall document.
7. Requirements are not entirely consistent in wording.
8. Further detail the preconditions in the Use Case 1
9. Fix minor inconsistencies within the diagrams to keep everything the same
10. Reinterpret sequence diagrams created for the use cases for return-call architecture
11. Update executive summary to represent further additions
12. Context diagram missing technician
13. Requirements are not consistent with the code implementation.

All review issues have been resolved for the final release as of May 9, 2020.

Raised in Coding

1. Values running past zero and becoming negative when adjusting speed.
2. Visual issues in terms of centering when trying to format each button, background, and included images.
3. Within adjustThrottle, we were having issues calculating the difference using the bridge array.
4. adjustThrottle changing throttle too slowly resulting in the current speed to exceed to the desired speed.

5. Rounding error with desired speed and current speed, number was too large for visual.
6. Trouble maintaining the desired speed, fluctuating too much.
7. Fix the default of the engine, we had it as “on” at all times.
8. Once we figured out how to stabilize the current speed, the current speed was always too much lower than the desired speed.
9. We realized we had not created any sort of emergency scenario/backup so we had to revisit our code to include it and the according features.

All coding issues have been resolved for the final release, as of May 9, 2020.