Cruise Control Software Development Version 0.05

Eric Altenburg, Michael McCreesh, Hamzah Nizami, Constance Xu

Team Mike

Stevens Institute of Technology

CS 347 — Software Development Process

We pledge our honor that we have abided by the Stevens Honor System.

Contents

Executive Summary

Team Mike is a startup initiative aimed at solving problems that come to light as society begins to adopt new technologies. One of which pertains to autonomous driving and "smart cars" as they are beginning to break into the automobile market more and more every year with examples such as Tesla and Ford. In a perfect world, if every driving car were to be a smart car with "autopilot," then it would make sense for each of them to communicate cruise control data with each other to reduce traffic build-up and make traveling more efficient. Through rotational leadership on a monthly basis, each team member possesses a set of distinct skills that can translate to highly efficient work sessions. Aside from developing a traditional cruise control, the aim is to make the software open source as it will serve as the foundation to a more interconnected logistical future in which autonomous cars can fully achieve their potential within a growing technologically advanced society.

1 Introduction

The past century has seen an explosion of innovation on a scale never before seen in human history. The rapid development and refinement of a myriad of motor technologies catalyzed the innovation process by allowing ideas to transfer at rates never before seen. With the constraint of distance loosening thanks to every iteration of the automobile, people have been granted more freedom to do what they please. While the automobile has enhanced society in several ways, there are some glaring problems that need to be addressed to continue the current rate of human progress and innovation. One of the biggest problems is driver fatigue, which is responsible for approximately 72,000 crashes annually. Programming the automobile to be reliably autonomous to a degree is one way to circumvent the issue of driver fatigue and making roads safer. That's exactly what cruise control aims to do. By moderating the speed of the vehicle by itself, cruise controls aim to lessen the effects of driver fatigue on the road. However, the technology is not perfect, and Team Mike aims to resolve that by creating the perfect cruise control system.

When it was implemented in 1958, cruise control was suitable for that era, however, as technology continually improves, the embedded software must as well. For longer car trips, it does not make sense for users to constantly accelerate and decelerate for several miles; this is where cruise control comes in. By being able to set the speed, the user is now able to accurately set and maintain the speed they wish to without having to

constantly intervene which, over time, would cause less harm to the vehicle as variable speed and RPMs are not as desirable as that of near-constant.

Our group intends to use an Agile method to implement our version of cruise control. This would encourage code sprints in two-week lengths where work will be divided into smaller sections and distributed based on each members' strengths. The Agile method may vary as to which one we specifically choose but regardless, this project will not come to fruition using Waterfall or other methods. The way that we are going to organize our group is through weekly meetings and ensuring that everyone knows their tasks for the week. These weekly meetings can also serve as a place where team members mention any obstacles that have come in their way when trying to resolve a problem, and also serve as a time where the team as a collective can try to think of solutions around those obstacles. We also plan on giving real-time updates about any progress made through Slack in order to ensure all members of the team are equipped with the most recent developments of the project. We also intend on using Java or C++ to implement our cruise control system. We believe that the high-performance Java provides and the fact that its part of the Object-Oriented Programming (OOP) paradigm makes it perfect for embedded systems.

There are a plethora of features that make up cruise control. As most cruise controls are seen on vehicles, there are generally a few buttons: increase speed (and sometimes decrease speed) and the button that starts cruise control. Once cruise control is set, the user then has the ability to increase, decrease, or maintain the speed of a vehicle. Furthermore, if the user presses on the brake, then cruise control is automatically deactivated. The system also allows for the operator of a vehicle to manually deactivate it.

To successfully implement the features of maintaining a steady speed and a safe distance away from other cars, there are certain requirements to fulfill. At a high level, proper hardware with fast, reliable sensors is critical for this project. Furthermore, lives are fundamentally dependent on this product and therefore it requires software that is error-resistant and thus a strong development and QA (quality assurance) team. In addition, the software must also perform it's expected tasks of moderating speed, maintaining a safe distance and switching back control to the driver when prompted to quickly. Having a cruise control software that is slow would be ineffective, so speed and accuracy are critical requirements for the software as well as the hardware. This is a mission-critical system because if it were to fail, it would put the lives of people in danger.

2 Requirements

2.1 Input

1. The power button that allows for the state of the system to change from on to off or vice versa.

2. When pressed, a button will either accelerate or decelerate in 1 mph increments.

3. When the brake is pressed, the cruise control system will unset the speed and give control back to the user until the user specifies a new speed to be set.

4. If the entire car turns off, then a signal will be sent to turn off the cruise control system.

5. RPM (Rotations Per Minute) sensor should be connected to the front axle and able to take readings for accurate speed calculations.

6. Engine sensor must be able to take input from the engine to tell the cruise control system to turn on or off.

    6.1. If the engine is turned on, the cruise control system must be ready for use within 3 seconds of the engine being turned on.

7. If the gas pedal is pressed, the vehicle will continue to accelerate at the control of the user, but when the gas pedal is released the cruise control system will continue to the previously set speed.

2.2 Output

1. Keep a log of activity in the system files to help debug in the event of a malfunction.

2. For every speed increase made by the user in the cruise control system, a visual indication by the software is necessary. This is so that the user minimizes their own human error. So, every time you increase the speed, you can see the cruise control speed on the car menu increasing by however much the user wants it to.

3. The system displays successful activation or deactivation in 10 milliseconds.

3.1. Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.

4. Keep a log of every time the cruise control system is used and at what speed written to the system files.

5. Once the cruise control system is turned on, it must be readily available. The engine sensor must be able to understand that the engine is on and tell the cruise control system that, if the user so wishes, it must activate.

6. With all the inputs it is taking from all the sensors, the software must be able to deliver the desired output for each function in less than 15 milliseconds.

   6.1. The brake pedal sensor must be able to tell the software that the user has stopped, and stop the cruise control system. *Numbers are subject to change depending on how inputs are received from surrounding system, but are expected to be in around that same ballpark.*

2.3 Functional

1. While the cruise control system is on and a speed is set, it must be able to increase and decrease the speed by 1 mph.

2. Able to switch the cruise control system on or off provided the engine is on.

3. Interpret engine on and off signal to allow for the cruise control system to be turned on within 3 seconds of received the signal.

4. Only allow the cruise control system to be activated when at a minimum speed of 25 mph.

   4.1. If the speed is set to 25 mph, the user cannot decrease the speed below 25 mph.

5. Maximum speed that cruise control system can be set to is at 125 mph.

6. While the cruise control system is on, if a user is driving at a speed of at least 25 mph and decides to set the speed, the cruise control system will maintain that speed.

7. If the user presses and continues to press the brakes, the user will not be able to set the speed.

8. If the user presses and continues to press the gas pedal, the user can set the current speed for the cruise control system, but it will immediately pause as stated in the input requirements then resume after the user releases the gas pedal.

2.4 Security

1. No external interface to reduce potential tampering. This applies to both the hardware such as sensors and the general cruise control system software. There would be no easy access to the cruise control system hardware or software so that the likelihood of someone being able to create issues is reduced substantially.

2. No Internet or Bluetooth connection. This is so that no bad actors can meddle with the car's system and hence, keeps the drivers safer as cyber security becomes more of an issue.

3. An administrator will need hardware to make changes to the mechanisms. The administrator must be authorized to make these changes and they must be a trusted third party or those who created the cruise control system themselves.

3 Requirements Analysis Model

3.1 UML Use Cases & Diagram

1. Use Case 1: User activates cruise control system

    1.1. User requests an activation of the cruise control system.

    1.2. Cruise control system provides a visual feedback that it is ready for activation.

    1.3. Cruise control system activates.

2. Use Case 2: User sets the speed

2.1. Cruise control system requests values from sensors.

2.2. Sensors provide values for approval to set cruise control system at current speed.

2.3. Cruise control system requests the Engine Management System (EMS) set the speed at current position.

2.4. EMS Speed (Throttle) is set at current speed.

2.5. Cruise control system provides visual feedback to the user that the cruise control is set and working.

2.6. Sensors provide the changing environmental information to the cruise control unit (such as speed, request for increase/decrease speed, and brake).

2.7. Cruise control system detects the changes from the sensors and request adjusting speed or deactivating Cruise Control system accordingly.

2.8. Speed (Throttle) position is continuously set to new values to ensure that the speed remains constant.

2.9. Speed is continuously reported to the cruise control system.

3. Use Case 3: User adjusts speed

3.1. User requests an adjustment of cruise control system speed.

3.2. Cruise control system provides visual feedback that the system will alter the speed of the vehicle.

3.3. Cruise control system slowly adjusts the speed of the vehicle to match that of the request.

3.4. When the desired speed is reached, the cruise control system will provide visual feedback that the adjustment has been completed.

4. Use Case 4: User deactivates cruise control system

4.1. User requests a deactivation of the cruise control system.

4.2. Cruise control provides visual feedback that it is ready for deactivation.

4.3. Cruise control system deactivates.

5. See Figure 1 for general use cases of cruise control system.

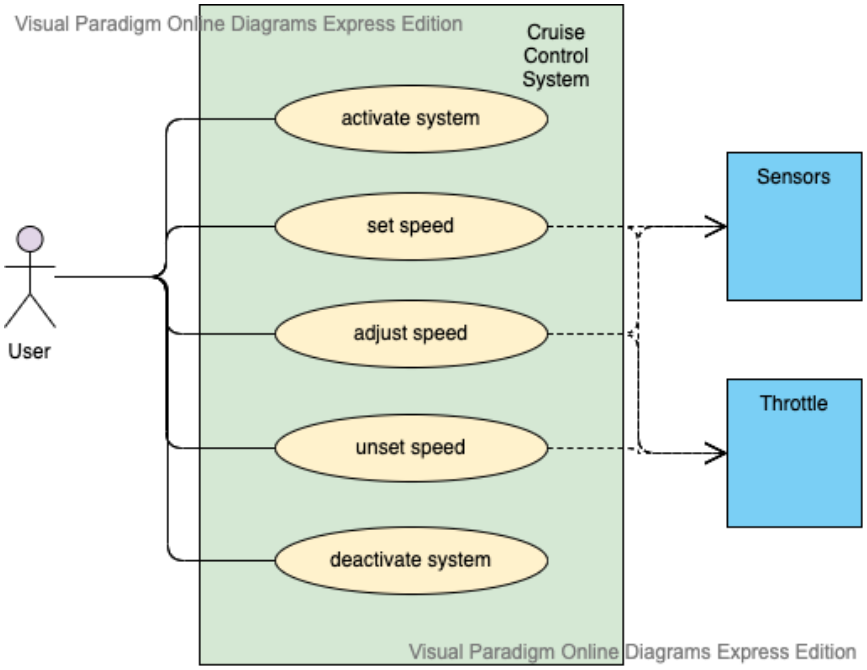


Figure 1: Sample UML diagram for cruise control system use cases.

3.2 UML Class-Based Modeling

1. See Figure 2 for the class-based modeling for the cruise control system.

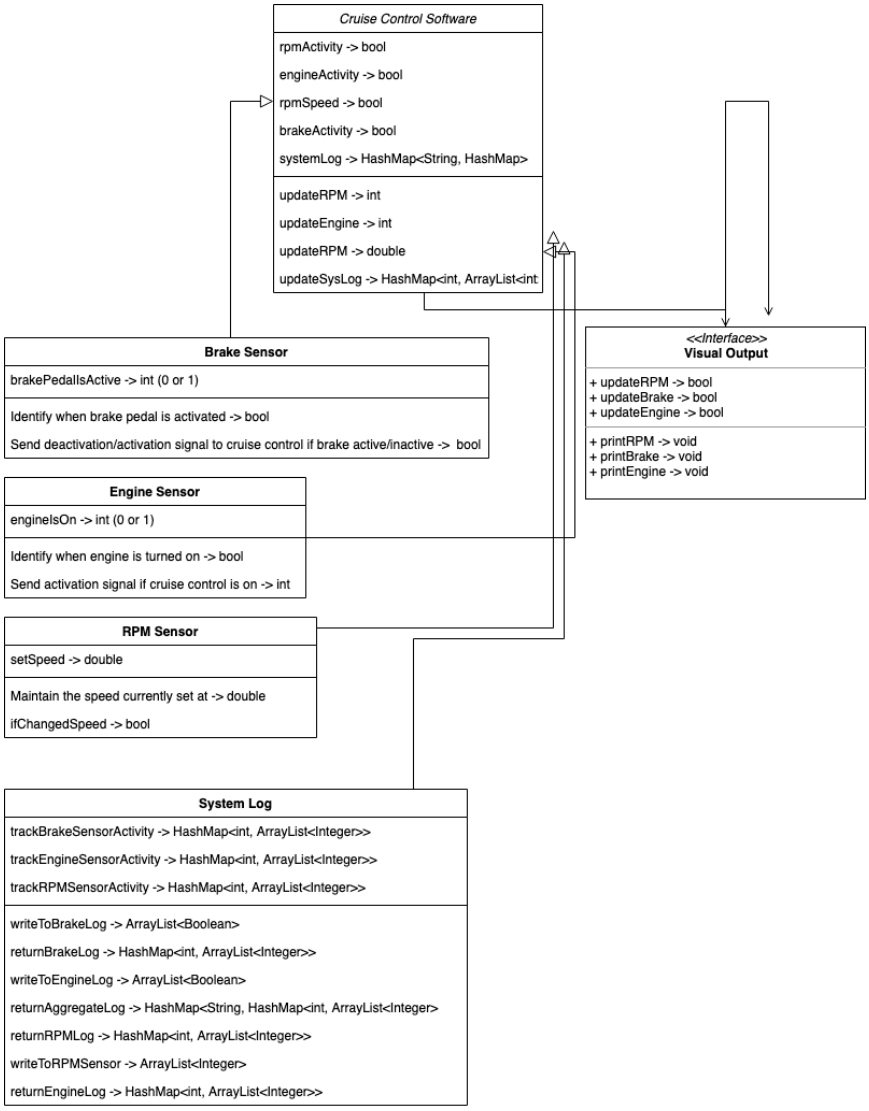
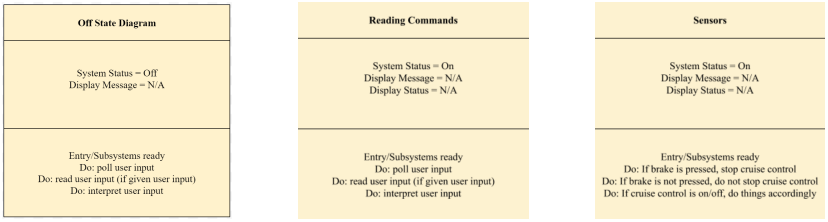


Figure 2: Sample UML class-based model for the cruise control system.

3.3 UML CRC Model Index Card

1. See Figure 3 for the CRC Model Index Cards for the cruise control system.



(a) 1a (b) 1b (c) 1c

Figure 3: Sample UML CRC model index cards for the cruise control system.

3.4 UML Activity Diagram

1. See Figure 4 for the activity diagram of the cruise control system.

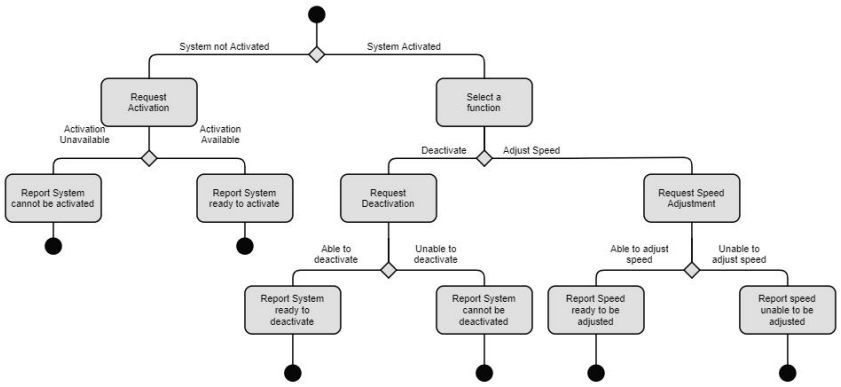


Figure 4: Sample UML activity diagram for the cruise control system.

3.5 UML Sequence Diagram

1. See Figure 5 for sequence diagram of cruise control system.

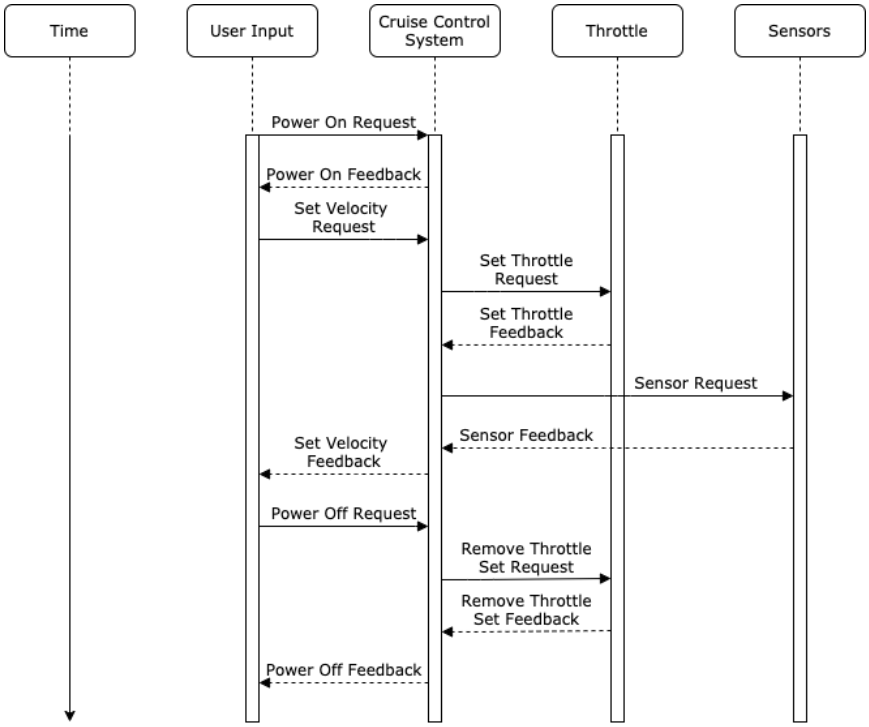


Figure 5: Sample UML sequel diagram for cruise control system functionality.

3.6 UML State Diagram

1. See Figure 6 for the state diagram of the cruise control system.

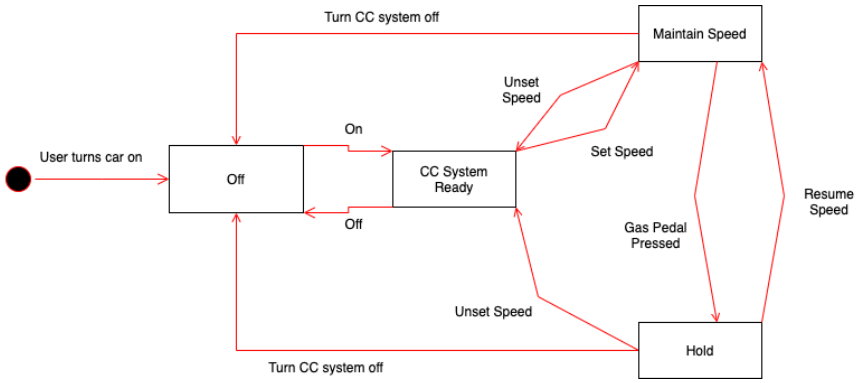


Figure 6: Sample UML state diagram for the cruise control system.