COMPSYS723 Assignment 2 - Cruise Control System In Esterel

Bailey Clague & Marianne Healey - Team 18

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand

Abstract

This report outlines our implementation of a cruise control system, in the synchronous programming language of Esterel. This was done through designing functional specifications of the system through context diagrams, functional analysis and state machines. Various states were looked at; on, off, standby and disable, and functions written in C were utilised, in order to translate our outlined functional requirements into a safety critical, reactive and concurrent executable system using Esterel.

1. Introduction

The controller designed and discussed in this report has been designed to implement the functionality of a cruise control system seen in many modern cars today. Esterel was utilized for this task as this language is commonly used to model real-time systems, due to its synchronous and concurrent nature. Models of our system were created initially using finite state machines (FSMs) and context diagrams which were then translated and implemented in code, emulating a real cruise control system.

Outlined throughout the report is the systematic and illustrative breakdown of the functionality of a cruise controller. This includes: an analysis of inputs and outputs, the processing of these inputs to appropriate outputs and an overarching state machine for the system. This is followed by looking at how Esterel was specifically implemented in the system by examining the language's synchronous and concurrent nature. The testing and verification of our model is discussed next and subsequently a brief summary of our design and implementation will be mentioned to conclude this report.

2. System Design

The cruise control system's basic functional requirement is to keep a car's movement at a given speed, never varying above or below the given speed. This can be done without using the brake or accelerator pedals. For this system we have defined several values in order to create a controlled environment for testing and evaluating functionality.

2.1. System Constants

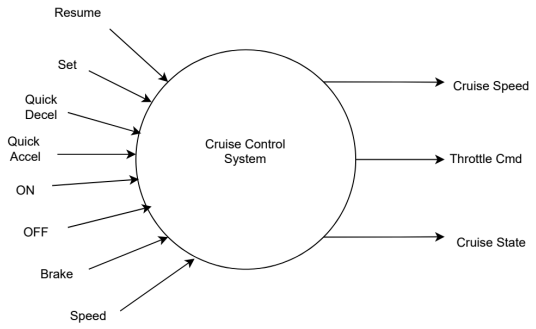
The following parameters have been defined in our system:

For the proportional-integral controller, the Kp value is 8.113 and the Ki is 0.5. When the cruise control is active the PI controller will be used to maintain the desired set speed.

Other constants include a speed minimum and maximum of 30 kilometers per hour and 150 kilometers per hour respectively. A speed increase of 2.5 kilometers per hour. A throttle saturation maximum of 45 percent and a pedals minimum of 3 percent to enable appropriate detection of pedals being pressed.

2.2. System Interface

The inputs and outputs of the cruise control system are defined below in the top level context diagram seen in Figure 1.



.

Figure 1: Top Level Context Diagram

The system overall has eight inputs and three outputs. The inputs include six pure signal inputs and three float valued inputs. The on and off pure signals are used to turn the cruise control on and off, whilst the set and resume are used to set a target speed and resume the cruise control after braking. The quick acceleration and quick deceleration pure signals are used to control the cruise speed. Accelerator, brake and speed are float values and are all inputs that have specific values that mimic the behavior of a vehicle.

The outputs of the system are two float values showing the cruise speed and the throttle and one enumerated value which is a string, describing the current cruise state. These cruise states include; on, off, standby and disable.

2.3. System Breakdown

Figure 2, seen below, describes the way the inputs and outputs interact with each other as they enter and exit the four loops within the system. The four loops are: the cruise control loop, the pedal control loop, the cruise speed loop and the throttle controller loop. Below will outline the design and functionality of three of the four loops. The cruise state control design will be outlined in the next section of the report with a finite state machine as it is our most critical loop.

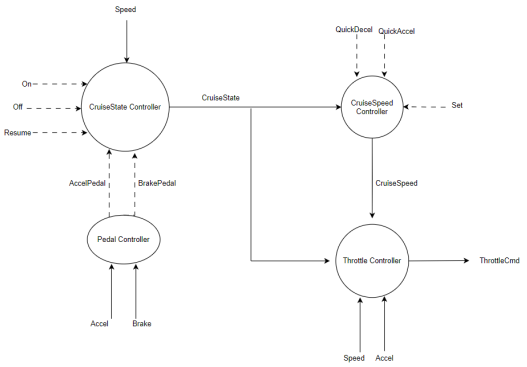


Figure 2: Low Level Context Diagram

2.3.1. Loop One: Pedal Control

The pedal control loop detects whether the acceleration and brake pedals are pressed. The inputs are Accel and Brake float values and the outputs are AccelPedal and BrakePedal boolean values. When either Accel or Brake values are above the PedalMin value of 3.0, AccelPedal and BrakePedal will be set to true, respectively. In any other case, AccelPedal and BrakePedal are set to false. These output values are then used in the State Control loop.

2.3.2. Loop Two: Cruise Speed Control

The cruise speed control loop controls the cruise speed of the cruise control system. It has four inputs: the cruise state, a string, and three pure signals, QuickAccel, QuickDecel and Set. This loop only has one output; the cruise speed, a float value. Cruise speed is only managed when the cruise control is enabled, when we are in ON, STDBY or DISABLE states. When the cruise control system enters the ON state initially, it will set the cruise speed as the current speed of the car. If the set button is pressed within the ON state, then the Cruise Speed can be changed again and will be set to the current speed of the car. If the QuickDecel button is pressed within the ON state, the cruise speed will be decreased from the current cruise speed by SpeedInc of 2.5 km/hr. If the QuickAccel button is pressed within the ON state, the cruise speed will be increased from the current cruise speed by SpeedInc of 2.5 km/hr.

If at any point there is an attempt made to change the cruise speed to a value less than SpeedMin, 30km/hr, the cruise speed is set to 30 km/hr. If an attempt is made to set it to a value greater than SpeedMax, 150 km/hr, then the cruise speed is set to 150 km/hr.

2.3.3. Loop Three: Throttle Control

The throttle control loop controls the regulation of the car speed. It has four inputs: CruiseState, read as a string, CruiseSpeed, Speed and Accel, floats. It outputs the Throttle Command, a float value. If the system is in Cruise State Off, the speed does not get regulated and is instead controlled by the accelerator pedal. If the cruise control is on, it is automatically regulated. The throttle control uses a PI controller to determine appropriate values given the current conditions. The implementation of this PI controller is described further in the implementation section below.

2.4. State Control

Figure 3 below describes through a state machine the overall state control of the cruise control system. The state machine below shows the design and functionality of the cruise state control loop seen in Figure 2. The diagram shows what conditions need to be met to transition from one state to another, as well as conditions that keep the system in their respective states. Our system has four different states; these are illustrated and further outlined below.

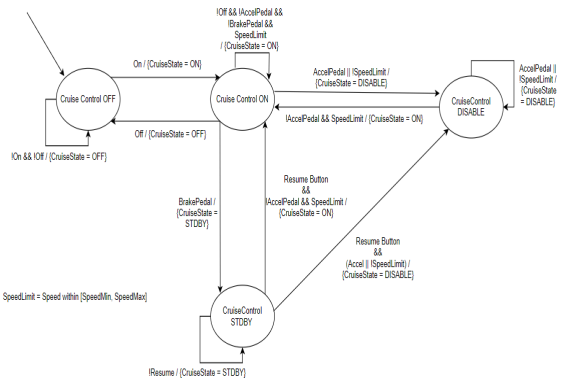


Figure 3: State Control Finite State Machine

2.4.1 Off state

Our system starts in the off state and will immediately transition into this state at any given time if the off input is high/pressed. In the off state all other inputs except for the on input will be ignored. When the on input is driven high the state control will output the state as "ON" moving into the on state, otherwise we will remain in the off state.

2.4.2 On state

The on state is the central state in our finite state machine as seen in Figure 3. In this state the cruise control will be responsible for regulating and maintaining the speed of the vehicle. From the on state we are able to move to the off state, standby state or the disable state. The system will remain in the on state whilst the accelerator and/or brake pedal is not pressed and the car is driving within the speed limit (not below speed minimum and not above speed maximum). If the brake pedal is pressed the system will immediately transition into the standby state where the cruise control is paused. If the accelerator pedal is pressed or the speed limits are breached, the system will immediately step into the disable state where the cruise control is disabled. If the off button is pressed in the on state the system will go back to the off state.

2.4.3 Standby state

The standby state, which is only accessible from the on state, is responsible for disabling the cruise control when the brake pedal is pressed. The vehicle will remain in this state until either the resume button is pressed, the vehicle is within speed limits and the accelerator is not pressed. In which case it will return to the on state. If the resume button is pressed and the vehicle is in breach of speed limits or the accelerator pedal is pressed it will transition into the disable state, disabling the cruise control altogether.

2.4.4 Disable state

The disabled state is reachable from the on and standby states. The system remains in the disable state whilst the accelerator is pressed and the speed limit is breached. Once the accelerator is no longer pressed and the vehicle is within the speed limit the car will return to the on state, where the cruise controller will maintain the speed of the car at the target set speed.

3. Implementation

Through translating Figures 1,2 and 3 into Esterel code we implemented a single module with four simple loops that concurrently handle appropriate inputs and give the desired outputs. The module also takes all inputs and constants already outlined above. Below will detail Esterel features used to implement the logic outlined above.

3.1. State Control Implementation

A present case statement is used to transition between states in the state control loop. A present statement is used to check the presence of the pure signals checked within each case statement. The four cases are: Off and not On, On, Resume and any other cases. Off, On and Resume are push button inputs which are pure signals. In the first case, Off and not On, the cruise state is simply set to OFF. In the second case, On, an if statement is used to check whether the on button is pressed from the off state or within the on state, only in this case will the cruise state change to ON, else it will remain unchanged.

In the third case, Resume, there is a nested if statement. First, it is checked whether the system is in STDBY state, if it is not the cruise state remains unchanged, if it is, another if statement is entered. Here there are three different if statements present, if the car is within speed limit and the accelerator pedal is not pressed, the cruise state will go into ON state, if the car is not within speed limit or the accelerator pedal is pressed, the cruise state will go into the DISABLE state. In any other cases, the cruise state will remain in STDBY state.

The final case deals with all the other cases and is also a nested if statement. First, it is checked whether the system is in the OFF or STDBY state as these states can only be entered and left if the off and resume buttons are pressed respectively. If the system is in either of these states, the cruise state remains unchanged, if the system is not in these states, the next if statement is entered. Here four different if statements are present. If the brake is pressed and we are not in the DISABLE state, the cruise state will go into STDBY state. If the speed limit is breached or the accelerator pedal is pressed, the cruise state will go into the DISABLE state. If the speed is within speed limit and neither the brake or the accelerator pedal are pressed, the cruise state will go into the ON state. In all other cases the cruise state will remain unchanged.

3.2. Pedal Control Implementation

The pedal control loop in our Esterel implementation is our simplest loop. We utilise an if statement for each the brake, and the accelerator pedal to see if either of the pedals has broken the 3 percent threshold. If it has, we emit the respective pedal has been pressed, else we emit false to show the pedal is not pressed. The case that both pedals are pressed in the same tick, we have given

priority to the brake pedal, so the system will respond as if the brake pedal has been pressed.

3.3. Cruise Speed Control Implementation

The cruise speed control follows functionality and behavior outlined in the cruise speed control section above. This behavior was implemented in Esterel using the inputs and outputs shown in Figure 2. By using a present case statement in Esterel we were able to capture the pure signals; On, Set, QuickAccel and QuickDecel. Based on each one of these cases of the input being driven, we could then determine if the speed was within the given limits and set the cruise speed appropriately using an emit statement. In the cases where we didn't need to set a new cruise speed, a pre operator was used to set the new cruise speed to the previous.

3.4. Throttle Control Implementation

By taking advantage of several Esterel features we were able to translate the lower level context diagram seen in Figure 2, specifically the throttle control, into Esterel code. In order to control our throttle correctly we used a PI controller. For our project this was implemented in C code and then using Esterel we could call this function [regulatThrottle()] using the speed, cruise speed and a flag showing the previous state was on or not. The calls of this function were encapsulated within a if statement that took advantage of the "pre" functionality in Esterel, allowing us to identify the previous state before the current tick. The if statements, which follow the behavior outlined in the throttle control section, meant we could input the current speed and cruise speed, as well as set the state flag (isGoingOn) when needed to correctly maintain the vehicle's speed.

4. Testing and Validation

To test whether the code written was both deterministic and reactive as well as complying to the brief specifications, many test cases were run.

For a system to be deterministic it must always be entirely predictable based on its initial state and inputs, So, if it is given the same initial conditions and inputs, this system will always produce the same result.

For a system to be reactive, it should be able to react to any inputs given. In the context of this system, this means that regardless of the values of the six inputs, an output must be able to be produced.

Using the vectors in and comparing the results of our system to the vectors out file, we were able to verify the functional correctness of our controller. Using syntax and error detection by the compiler when building both the .xev and .xes, as well as the signal checker and input/output GUI we were also able to ensure the function correctness of our system. Although the test cases provided were limited, the addition of our own also allowed us to find and fix flaws in our design.

The use of Esterel features such as the "pre" and "await" keywords in our program allow us to prevent cyclic dependencies and therefore avoid causality errors in the running of the system.

Although we have no way to fully verify our system with the tools provided, we have ensured, to the best of our ability, the functional correctness, determinism and reactiveness of our system.

Instructions on how to run the system in Esterel, with both the .xes and .xev can be found in Appendix A and in the readme.txt file attached to the project.

5. Conclusion

This report detailed how a cruise control system was successfully implemented using the synchronous programming language Esterel. Detailed functional specifications were utilized to create finite state machines and context diagrams to model the system. The implementation of this system demonstrated Esterel's effectiveness in creating a reactive, safety-critical, and concurrent system. The testing and verification confirmed the system's reliability to follow the project specifications and its reactive and deterministic nature. This project highlights the suitability of Esterel for real-time automotive applications and the importance of thorough design and testing in developing safety-critical systems.

6. References

Appendix A - readme.txt

INSTRUCTIONS TO RUN:

Open a terminal in the 'Cruise-Control-Files' directory

Clean and build both the .xes and .xev by entering the following commands

For the .xes:

```
make clean cruiseController.xes
./cruiseController.xes
```

The GUI where inputs and outputs can be displayed should open.
Change the font to huge to get a good window size.

Select the command panel and select "Keep Inputs"
Modify input values as needed and click tick when updating of the outputs is needed.

For the .xev:

make clean cruiseController.xev
./cruiseController.xev

select signals with the appropriate colour and click "apply" when ready to check outputs