COMPSYS 723 ASSIGNMENT 2 REPORT

Cecil Symes (csym531), Nikhil Kumar (nkmu576)

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

Abstract

A description for a vehicle cruise controller was given in the form of a specification list with certain design requirements. A cruise controller system was developed in Esterel and C, with Esterel controlling the control logic and control flow of the system, and C handling all data manipulation. The implemented system is described in detail in this report, with diagrams to aid understanding. Screenshot proof of the system outputs are also provided to prove functionality as requested.

1. Introduction

Cruise control systems are present in a large number of modern-day vehicles, notably consumer cars. Cruise control was developed in order to help reduce driver fatigue when driving long distances. Cruise control does this by automatically maintaining a desired speed, allowing the driver to rest their feet instead of having to constantly manage the driving pedals.

Cruise control is typically only meant for driving in straight lines at a constant speed, namely for highway or motorway driving. As such, the provided system can dynamically detect the speed of the car and any pedal inputs from the driver, allowing it to engage and disengage quickly and easily. The system must be reliable otherwise error can lead to injury or death.

2. Specification

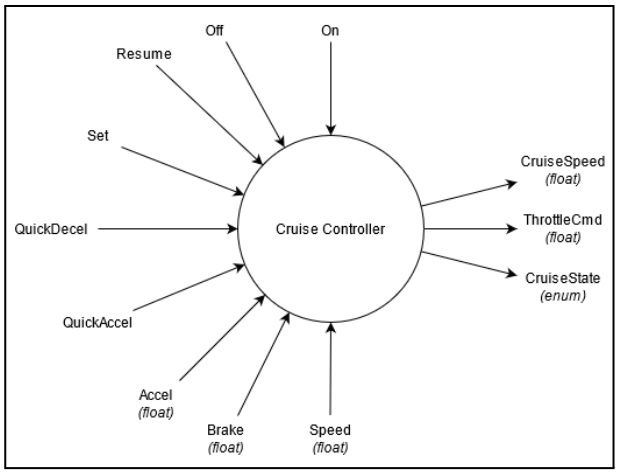

Figure 1. Context Diagram for the Cruise Controller System

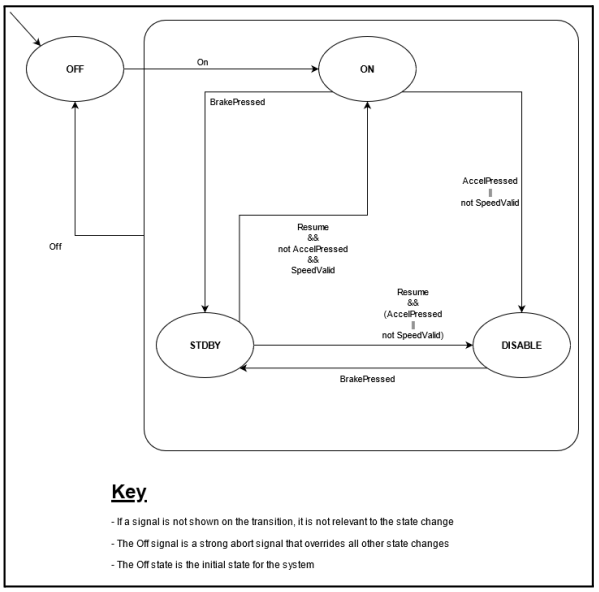

Figure 2. Finite State Machine for the system

The system overall has nine inputs and three outputs. Six inputs are pure signals, and 3 are float values. Of the outputs, two are floats and one is an enumeration, or enum. Figure 1 shows the context diagram for the cruise controller system.

Figure 2 shows the Finite State Machine used for the cruise controller system. The Off signal was given the highest priority, meaning that all states would return to the Off state when the Off signal was present, regardless of all the other signals.

3. Design & Implementation

3.1. Esterel Implementation

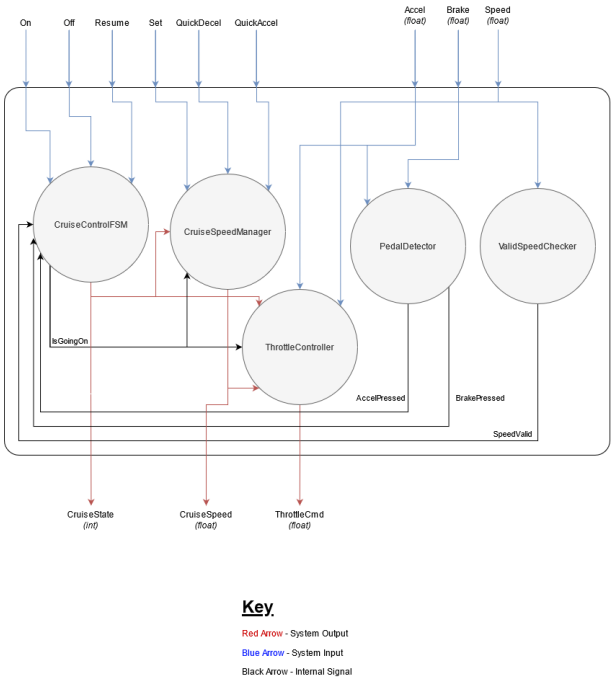

Figure 3 shows the functional diagram of the implemented cruise controller. Functionality was split as much as was thought to be reasonable, and that also suited the synchronous nature of Esterel. PedalDetector and ValidSpeedChecker both perform single functions, whilst CruiseSpeedManager, ThrottleController, and CruiseControlFSM are all more complex modules.

ValidSpeedChecker simply checks the current Speed each tick, and determines if it is within the valid range between SpeedMin and SpeedMax. If it is, then SpeedValid is emitted within the system.

PedalDetector also does a single function, and checks each tick if the current Brake and Accel values are above the PedalsMin threshold. If they are, then the respective signal is emitted to the system indicating that the pedal has been pushed past the threshold.

CruiseControlFSM is the module that is solely responsible for changing the state of the system. Transitions are taken as described in Figure 2. The CruiseState is emitted after all possible inputs for the current state are checked, meaning that state transitions are always detected and emitted instantaneously. In the implementation, CruiseState is emitted as an integer, as it was unknown as to how to use enumerations. The corresponding states for the cruise controller and the associated integer are as follows:

0 - OFF

1 - ON

2 - DISABLE

3 - STDBY

CruiseSpeedManager is the only module that is able to change the CruiseSpeed output. It will set the current speed as the target CruiseSpeed if initially moving from the Off to On state. Otherwise, it will constantly check for the Set, QuickAccel, and QuickDecel inputs. Set has the highest priority of these three inputs, meaning that if Set is pressed, then the current speed will be set as the target speed, irrespective of the other two inputs. QuickAccel and QuickDecel have no inherent priority, meaning that if they are both pressed in the same tick, the CruiseSpeed will not change at all.

ThrottleController is the only module that is able to control the ThrottleCmd output. If the system is in any state except for On, then the ThrottleCmd is simply tied directly to the Accel input. In the On state, the ThrottleController will use the regulation algorithms provided to calculate a ThrottleCmd output.*Headings*

3.2. Design Assumptions & Justifications

3.2.1. CruiseState is Integer Type, not Enum

As previously mentioned, getting enumerations to work in C and in Esterel was difficult, and so as a result integers were used instead to represent the current state of the cruise controller system. No functionality was lost as a result of this change, so it was viewed as justified.

3.2.2. No priority with QuickAccel and QuickDecel

The assignment brief did not specify whether there was an innate priority to the two inputs signals. Logically, if the two keys were pressed in the same tick, it would be equivalent to adding SpeedInc, and then immediately removing it, or vice-versa, meaning that there would not be an overall change in speed. As a result, the decision was simply made to ignore both inputs if they were input in the same tick. It was felt that this was a reasonable assumption, as any end user would not press both buttons at the same time and expect the CruiseSpeed to change meaningfully.

3.2.3. Set takes priority over quick accel and decel

Once again this was not outlined in the assignment brief, so it was assumed that Set would take priority over QuickAccel and QuickDecel, as this would not impact the system behaviour much.

3.2.4. Assume throttle ties to accel directly when cruise controller is not active

When in the On state, the Cruise Controller uses the provided C functions to calculate a ThrottleCmd amount. When in the other three states, it was assumed that the Accel value would be output directly to the Throttle value. This assumption was made because if a user were to turn off their cruise controller, then they would expect their throttle to be directly controlled by the accelerator pedal. As for the other two states, if the user were to brake or slow down below the minimum speed, then once again they would expect to have full control over the throttle of their car.

3.2.5. Use of C Functions and Procedures and placing of all constants in C code

Initially, the implementation consisted of all the constants dispersed throughout the Esterel code. This had issues however, as one constant could be defined multiple times in different modules, making it difficult to universally change a single constant. This also meant that Esterel had some data handling portions. The decision was made to move all the constants into the provided C file, meaning that they were all located in one place, making it easy to modify system parameters. This also meant that all data-handling was handled in the host language, C, and Esterel was left purely to handle the control logic.
Functions and Procedures were used as close to their intended purpose as possible, as defined in the EsterelPrimer. This meant that Procedures did not return any values, and only modified arguments provided in the left brackets, and Functions could return values and did not modify any provided arguments.

3.2.6. Braking in Disable moves system to Standby

The Esterel implementation uses an if statement to check for the current state and any state changes. This means that if the cruise control is in the Disable state and the brake is pressed, we would want to move directly to Standby. This is implied within the assignment brief, and as such a transition to handle this was added in the finite state machine.

4. Testing

System functionality was thoroughly tested, and two of the test cases are documented below. Screenshots are shown in chronological order of the ticks occurring.

4.1. Test Case 1

- Ensuring that QuickAccel and QuickDecel do not change CruiseSpeed outside of SpeedMin and SpeedMax
- Ensuring that ThrottleCmd is a reasonable value

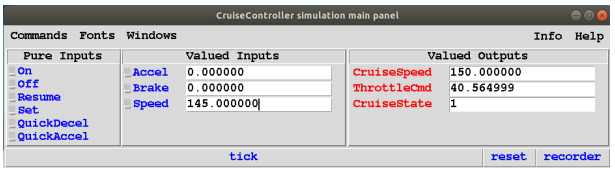

Figure 3. Tick 1 for Test Case 1

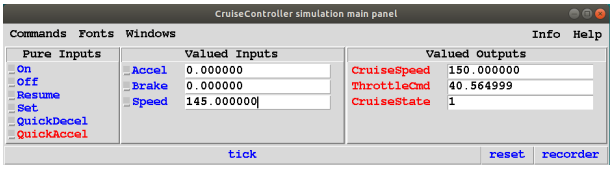

Figure 4. Tick 2 for Test Case 1

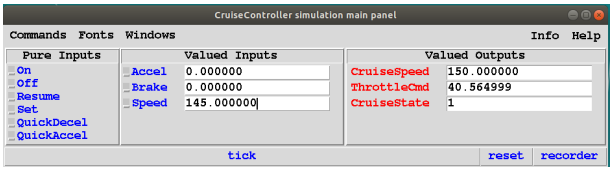

Figure 5. Tick 3 for Test Case 1

To confirm that the ThrottleCmd is reasonable:
Kp is 8.113 and Ki is 0.5. The difference between CruiseSpeed and Speed is 5, so assuming that Ki is 0, 8.113 * 5 gives 40.565, which is correct as shown above.

4.2. Test Case 2

- Ensuring that Resume moves the system from Standby back to the correct state

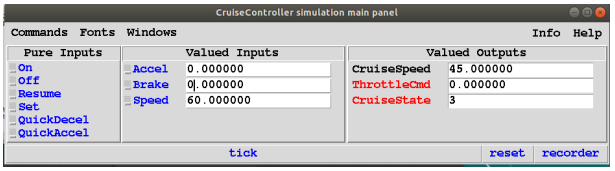

Figure 6. Tick 1 for Test Case 2

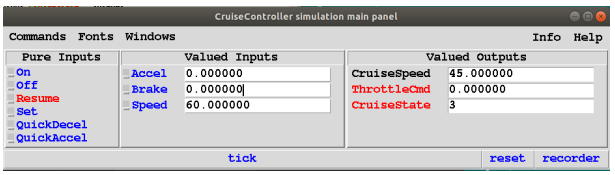

Figure 7. Tick 2 for Test Case 2

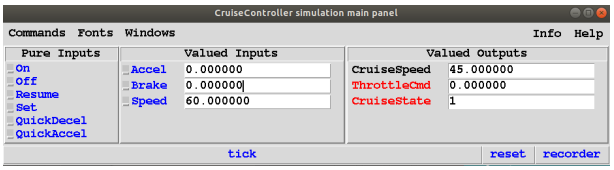

Figure 8. Tick 3 for Test Case 2

The Accel and Brake pedals are both below the PedalsMin threshold, and the Speed of 60 is between SpeedMax and SpeedMin. As a result, we can see that the system moves back to the On state when Resume is pressed, as expected.

Here is a short list of the tests conducted, based on the specifications given.

Condition	Result
Cruise off when other buttons pressed	True
CruiseControl on when on pressed	True
Cruise Control turned off when off pressed	True
Cruise Control turned on within speed limit, acceleration not pressed	True
Cruise Control is disabled when Acceleration is on	True
Cruise Control is disabled when the speed is over max speed	True
Cruise Control is disabled when the speed is over min speed	True
Cruise Control goes immediately to standby when brakes pressed	True
Transition to disable from Standby if speed valid, and resume pressed	True
CruiseSpeed is only managed when cruise control is not off	True
CruiseSpeed set to current speed when on or set is pressed thereafter	True
Cruise Speed increased by SpeedIncrease when QuickAccel is on, with bounds checking	True
Cruise Speed Decreased by QuickDec with bounds checking	True
Cruise Speed is set to speed when set is pressed with bounds checking	True

5. Time Commitment

	Project Setup	Writing Code	Testing	Report
Nikhil	1	5	4	4
Cecil	1	7	4	4

All times are in hours. Totals are approximately 30 hours combined, with 16 hours for Cecil and 14 hours for Nikhil.

6. Conclusions

Overall, a Cruise Controller system was designed and implemented in Esterel and C. All the control logic was handled in Esterel, and all data handling was left to the host language C. Some design assumptions were made, but none affected the overall functionality of the system by a large amount. The reliability and error free operation of the system is paramount, in any design of a cruise control system, due to its interfacing with the core vehicle features. We believe we upheld that requirement of the design.