COMPSYS 723 ASSIGNMENT 2 REPORT

Reeve D'Cunha (rdcu227) and Prabhnandan Singh (psin546)

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

Abstract

Here we present our report for the 2nd Assignment in COMPSYS 723. For this Assignment, we were tasked with developing a Car Cruise Control system using Esterel. Within this report, we will discuss the system on a high level, as well as providing a detailed discussion on each module.

1. Introduction

Cruise Control systems are ubiquitous in modern Cars and we were tasked with developing a simplified version. This purpose of this system was to maintain a constant speed for the Car, without pressing the accelerator or brake pedals. There are also additional inputs (e.g. quick Acceleration input) and the system must be able to detect these, react accordingly, and then manage the outputs to reflect the new state of the system. While Cruise Control systems can be created using any language, we were tasked with developing one that was Synchronous and Reactive. Hence, Esterel was used.

2. Overall Structure

2.1. Context Diagram

Figure 1 illustrates the Context Diagram of our Cruise Control System. This Context Diagram helps describe the Input-Output behaviour of our system. The inputs to the system consist of some buttons, as well as sensor readings.

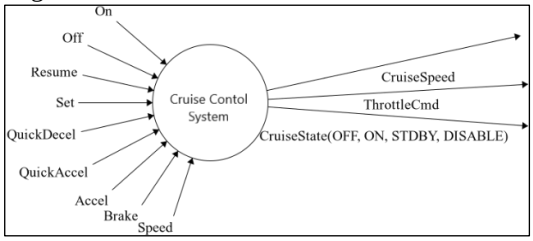

Figure 1: Context Diagram of the Cruise Control

On and *Off* enable and disable the system respectively. Similarly, *Resume* re-enables the system after it enters a standby period while braking. On *Set*, the system gets the current speed of the Car and sets it as the new default Cruise Speed. *QuickAccel* and *QuickDecel* increase and decrease the Cruise Speed respectively.

Accel and *Brake* are sensor readings taken from the Car's Accelerator and Brake pedals respectively. Finally, *Speed* is the Cars' current speed.

The system has three Outputs. First is the *CruiseState*, which is an Enumeration that describes the current status of the Cruise Control system. This can be either OFF, ON, STDBY, or DISABLE. *CruiseSpeed* is the current speed of the Cruise Control system, which can result in the Car either increasing or decreasing in speed, or remaining unchanged. *ThrottleCmd* accelerates the car as required depending on the speed of the vehicle, Accel pedal and the *CruiseSpeed*.

2.2. First Level Refinement

We've taken the top-level Context Diagram and have also produced a First Level Refinement, which further partitions our Cruise Control System. This is illustrated in Figure 2 below.

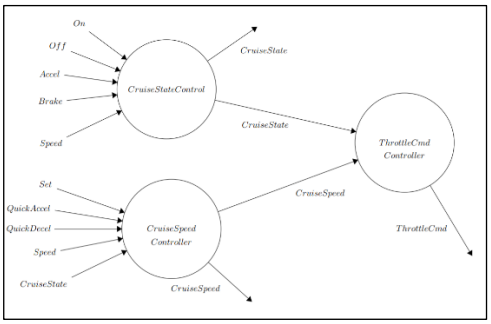

Figure 2: First Level Refinement of the Cruise Control

Here, we have split our system into three separate modules.

2.3. Finite State Machine

The refined context diagram has three units whose behavior can be depicted using finite state machines shown in figure 3, figure 4 and figure 5. Since these, modules run in parallel the combined FSM is shown in Appendix I. For readability reason, some of the conditions have been replaced with simpler words. These conditions are:

withinLimit = Speed > SpeedMin & Speed < SpeedMax
Accel = Accel > PedalsMin
Brake = Brake > PedalsMin

overMax = CruiseSpeed > SpeedMax
underMin = CruiseSpeed < SpeedMin
The values in the state circles are the values output in that state for the particular variable.

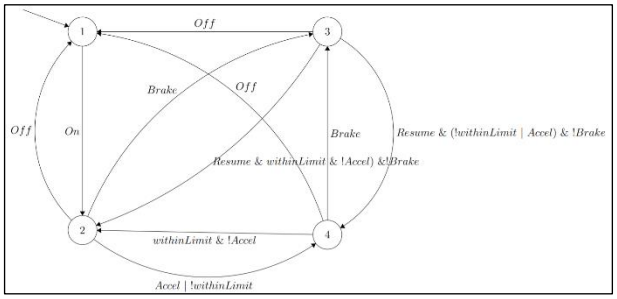

Figure 3: FSM Diagram: CruiseState

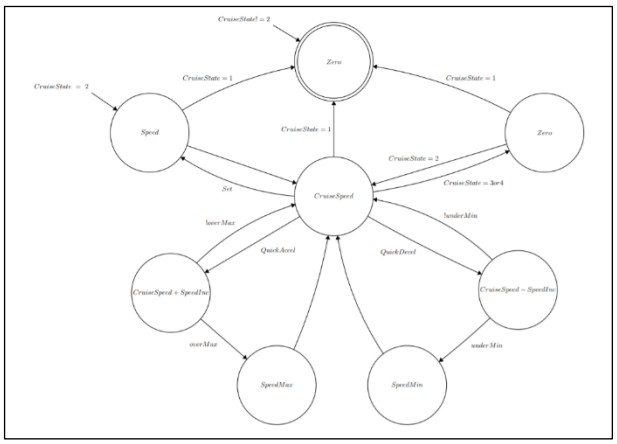

Figure 4: FSM Diagram: CruiseSpeed

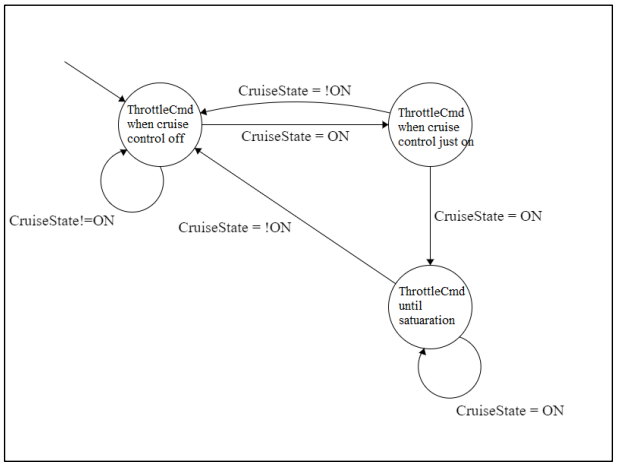

Figure 5: FSM Diagram: ThrottleCmd

3. Modules

The design of the cruise control in Esterel is based on the structure discussed in section two. There is one top level module based on the top level context diagram and three logic modules: *CruiseStateController*, *CruiseSpeedController* and *ThrottleCmdController*. These three modules run in parallel. More details on these modules is given below.

3.1. *CruiseControl*

This is our Top Level module. Its primary functionality is to run the other three modules in parallel and map the inputs and outputs within the modules and the external signals. The outputs are collected from the other three modules and is emitted as outputs by this module.

3.2. *CruiseStateController*

The behavioral requirements of our system is governed by the *CruiseStateController* module. Its primary functionality is to emit the ***CruiseState*1** output, which is referred to in the *CruiseControl* module for the ***CruiseState*.** The value of ***CruiseState*1** is determined by considering input and current state conditions. Additionally, detection of Pedal Pressing is also handled within this module and we chose not to abstract this functionality away i.e. use another module to detect valid pedal presses. An internal variable is used to keep track of the state during the state transition. A trap within a loop is used to determine the state so that the system is reactive and emits a signal every clock tick.

3.3. *CruiseSpeed*Controller

The primary functionality of this module is to emit ***CruiseSpeed*2**, which is mapped to ***CruiseSpeed*** in *CruiseControl*. This is the Car's speed as determined by the Cruise Control System. Within this module, we are also ensuring that the Cruise Speed is always maintained within the SpeedMax/Min thresholds that have been defined. Again, a variable is used to compute the *CruiseSpeed* value every clock tick. Since, it as unclear from the brief what the value of *CruiseSpeed* should be when the car is not being driven by the cruise control system, we decided to set the value as zero. We also used another internal signal offState which is used as the exit condition of a weak abort condition which is used to exit the inner loop where the *CruiseSpeed* is being controlled. This allows the inner loop to keep track of the *CruiseSpeed* even when the *CruiseState* moves to DISABLE or STDBY state. However, when the offState signal is emitted meaning the *CruiseState* is OFF, *CruiseSpeed* value is discarded.

3.4. ThrottleCmdController

The primary functionality of this module is to emit ***ThrottleCmd*3**, which is mapped to ***ThrottleCmd*** in *CruiseControl*. When the system is not ON, the car's speed is determined solely through the Accelerator Pedal Input. When it is ON, the speed is regulated, and this is done through the **regulateThrottle** function, written in C. This function, as well as its helper function

saturateThrottle use a PI controller to smoothly change the acceleration of the Car. The **saturateThrottle** function ensures that the throttle command is correctly saturated and that the acceleration is properly limited.

4. Conclusions

In conclusion, we designed a cruise control system based on the requirements given to us in the assignment brief. The system was implemented using Esterel V5, which efficiently allowed us to implement a reactive and functional cruise control system. We discussed some of our design decisions and how they lead to the final design in Esterel. We also carried out several tests, some of which are given in APPENDIX II. The results from the tests matched the expectations.

5. Work Distribution

	Contribution (hrs)	
Task	Reeve	Prabh
Design	3	3
Implementation	2	4
Debugging	1	1
Report	3	3

APPENDIX I

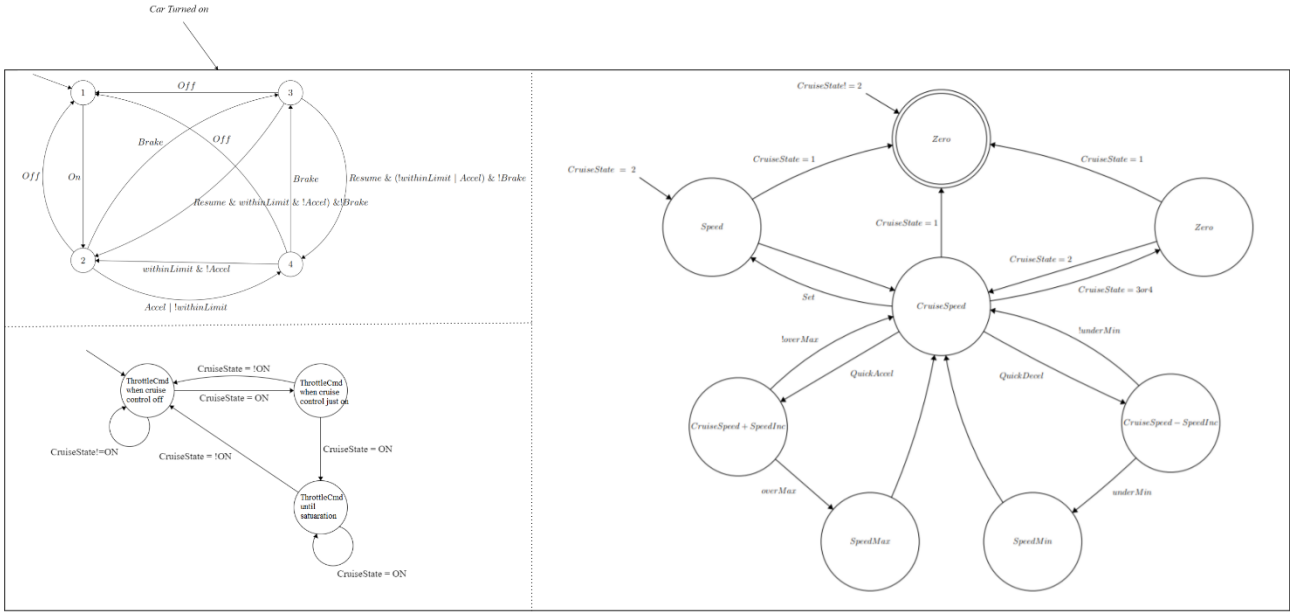

Figure 6: Complete FSM Diagram

APPENDIX II

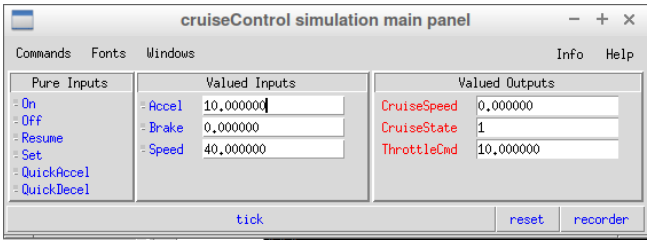

Figure 7: Cruise control OFF

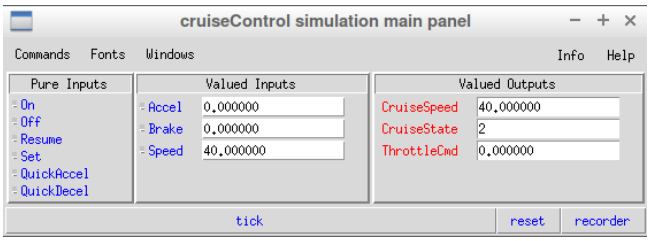

Figure 8: Cruise control ON

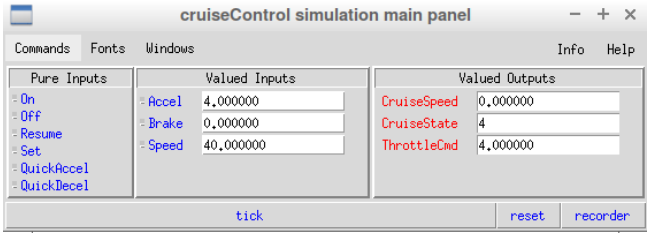

Figure 9: Accel pressed - Cruise control DISABLE

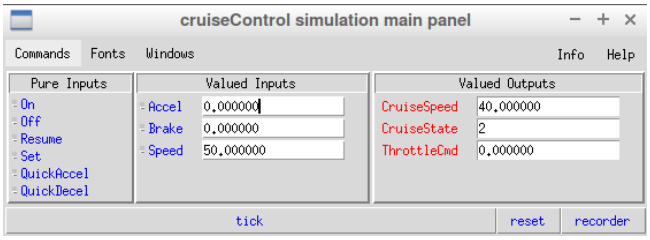

Figure 10: CruiseControl ON again - previous CruiseSpeed value used

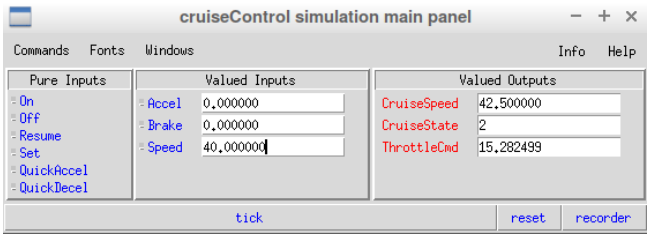

Figure 11: QuickAccel pressed