Cruise Control System utilising Esterel

Dylan Fu and Quentin Heng
Department of Electrical, Computer and Software Engineering
University of Auckland
Auckland, New Zealand

***Abstract*—We present the design of a cruise control system in the synchronous programming language Esterel. We developed a set of functional specifications for the cruise control system through data-flow diagrams and a FSM. We then implemented the specifications in Esterel along with some functions written in C. In doing this, we developed a reactive executable program that fulfils the specifications of the cruise control system. We then tested and verified our solution to make sure our system works as intended.**

I. Introduction

This report discusses the implementation of a cruise control system. The controller receives input from a vehicle's accelerator and brake pedals. It also receives input from control buttons that allow the driver to turn the cruise control on or off and set, increment and decrement the cruising speed. The system outputs the current state, cruise speed, and a throttle command for the vehicle's engine. The system uses a proportional and integral control system to regulate the throttle command. The cruise control system is developed using Esterel. Data handling, and PI controller functions are written in C. The developed system uses various synchronous programming constructs and features of Esterel. Concurrent state machines interact with the environment through interfaces, and with each other through signals for synchronisation and communication. This report will discuss the specifications of the system, our design solution, and verifying the system is causal and functionally correct.

II. Specifications

The main purpose of the cruise controller is to determine the automatic throttle response and cruise speed depending on the current state of the cruise controller.

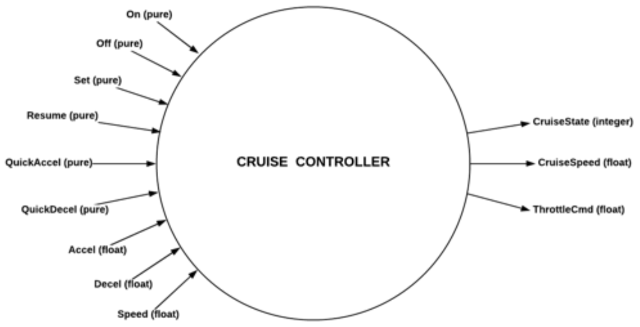


Figure 1: Top-Level Context Diagram

The overall behaviour, input and output of the cruise controller is described by the context diagram in. Figure 1. The cruise controller system receives inputs from the driver through the buttons (On, Off, Set, Resume, QuickAccel and QuickDecel) and pedals (Accel and Brake) of the car. The cruise controller receives the car's current speed as an input as well. Using these driver inputs, the system processes the input signals and produces three output signals (CruiseState, CruiseSpeed and ThrottleCmd) to be used by other components of the car to control the car.

A. *Inputs*

The on and off buttons are used to turn the system on or off respectively. When the system is on, the set button will set the current speed of the car as the desired cruise speed. The resume button is used when the cruise controller is in standby, this allows the system to re-enable the cruise control system with the previously set cruise speed, this allows the driver to resume cruising speed intuitively without having to reset the whole system. The driver can adjust the cruise speed while the cruise controller is on by pressing the QuickAccel and QuickDecel buttons to increment or decrement the current cruising speed respectively. If the accelerator and brake pedal are pressed when the cruise control is on, they will affect the state of the cruise controller, so that the pedal inputs can override the cruise control system. When the brake pedal is pressed the system will go into the standby state, once the brake and the driver presses resume, the system will leave standby and transition to either on or disable state depending on the speed and accelerator pedal. When the accelerator pedal is pressed the system will go to the disable state, once the accelerator pedal is released and the speed is valid, then the system will go into the on state. The current speed of the car is also used to check whether the speed is valid to be set as the new cruise speed. Consequently, this means that the system will be disabled if the car is outside the valid speed thresholds.

B. *Outputs*

The CruiseState is outputted so the driver knows whether the state of the cruise control system is in On, Off, Disable or Standby state. The CruiseSpeed is the target cruising speed, this is outputted so the driver knows what the current cruise speed of the system is, this feedback is needed to determine future adjustments. The ThrottleCmd is the output related to the throttle response (i.e. fuel flowing into the engine) needed to reach/maintain the target cruise speed.

III. Overview of Design

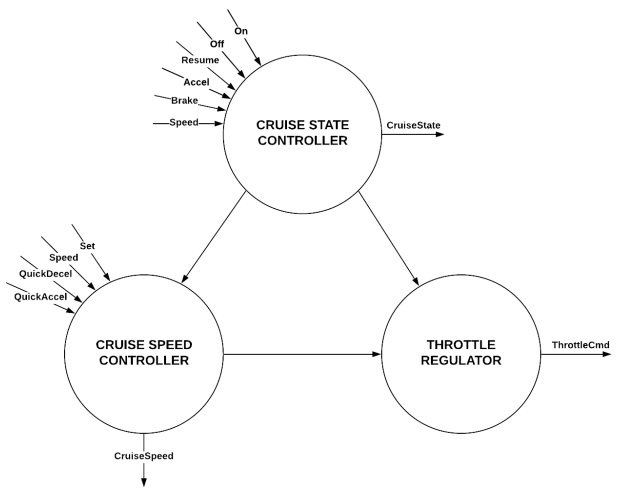


Figure 2: Lower-level Context Diagram

The top-level context diagram in Figure 1 can be further refined into a lower-level context diagram in Figure 2. The

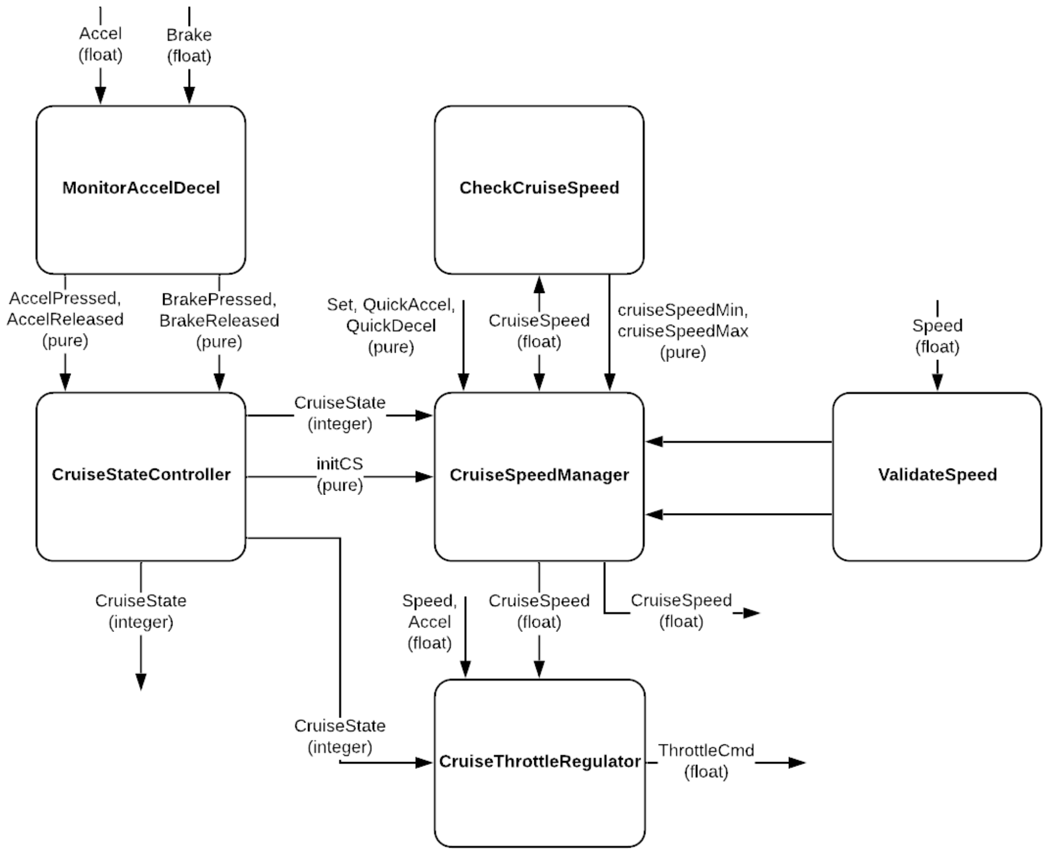
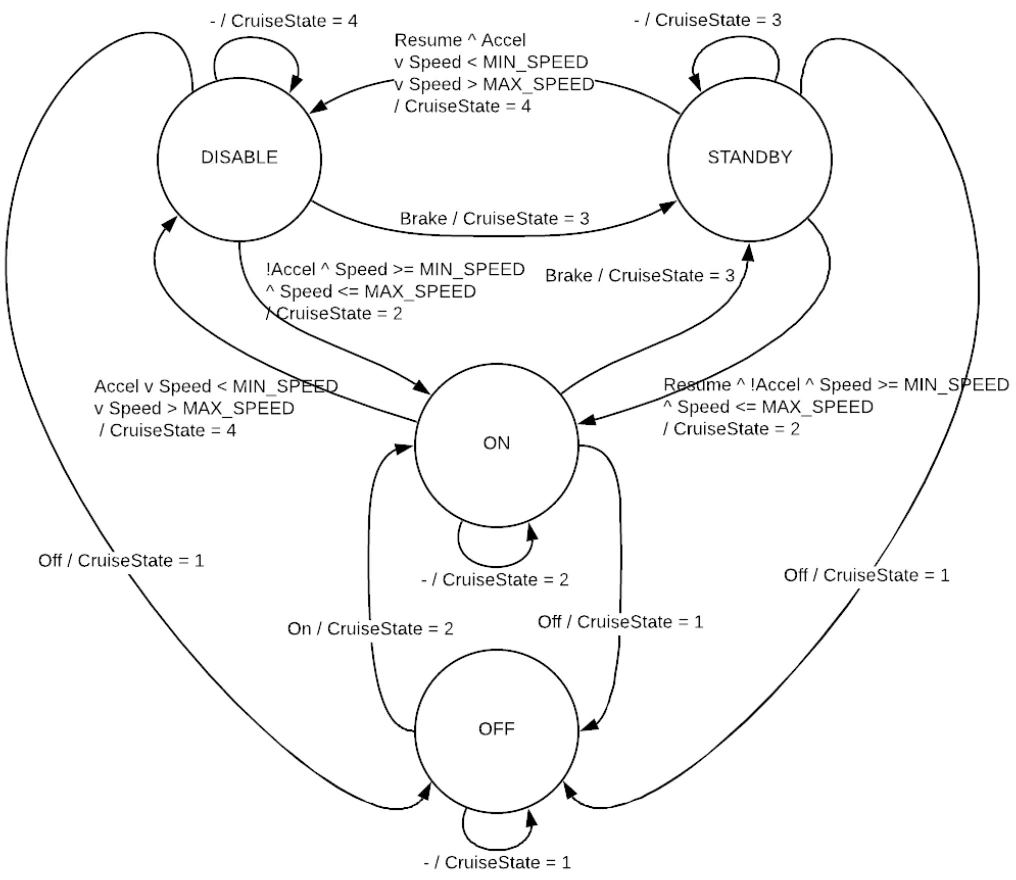


Figure 3: Final Data-flow Diagram



- / CruiseState = 4
- / CruiseState = 3
Resume ^ Accel v Speed < MIN_SPEED v Speed > MAX_SPEED / CruiseState = 4
DISABLE
STANDBY
Brake / CruiseState = 3
!Accel ^ Speed >= MIN_SPEED ^ Speed <= MAX_SPEED / CruiseState = 2
Brake / CruiseState = 3
Accel v Speed < MIN_SPEED v Speed > MAX_SPEED / CruiseState = 4
Resume ^ !Accel ^ Speed >= MIN_SPEED ^ Speed <= MAX_SPEED / CruiseState = 2
ON
Off / CruiseState = 1
- / CruiseState = 2
Off / CruiseState = 1
On / CruiseState = 2
Off / CruiseState = 1
OFF
- / CruiseState = 1

Figure 4: Mealy FSM for CruiseStateController

refined lower-level diagram emphasises the modules in the system and how each package is related with the incoming system inputs, interdependent signals, and outgoing output signals. The system is decoupled into three packages, which each handle the computation of a single output. The outputs are the cruiseState, cruiseSpeed and throttleCmd which are handled by CruiseStateController, CruiseSpeedManager and CruiseThrotlleRegulator respectively. These modules are dependent on the inputs to the system and the outputs of the parallel running modules due to the use of internal signals.

However, we decided to further decompose the modules by having three supporting modules (see Figure 3): MonitorAccelDecel, ValidateSpeed and CheckCruiseSpeed. The MonitorAccelDecel module handles the Accel and Brake floating point inputs and emits pure signals to be used in the CruiseStateController. The ValidateSpeed handles the conditional checking of the car speed and emits pure signals for the CruiseSpeedManager module. The CheckCruiseSpeed module handles the checking of whether the next increment/decrement of the cruiseSpeed is valid and produces pure signals to be used by the CruiseSpeedManager. We decided to produce pure signals to make the system easier to understand, modify, and especially easier to test and verify which will hopefully lead to less erroneous errors.

IV. Esterel Module Design

The overall cruise control system is decomposed into modules that are responsible for processing certain inputs of the system to produce the corresponding output. The main components of this design are the CruiseStateController, CruiseSpeedManager and CruiseThrottleRegulator. These components are further supported by supporting modules that process input signals to produce internal control instead. These modules are all linked together through the top-level CruiseController module that interfaces the signals of the modules.

A. *CruiseStateController*

The current behaviour of the cruise control system is determined by this module. The system will react to its external environment and then switch between the following states: On, Off, Standby and Disabled. These states are determined by processing the external On, Off, Resume input signals, and internal brake, accelerator, and speed control signals that are derived from their corresponding external inputs. This module is a Mealy FSM shown in Figure 4. The FSM was developed using Esterel state variables and traps to imitate go-to statements.

B. *MonitorAccelDecel*

This module is responsible for processing the external brake, and accel inputs from the system. They determine whether the brake or accelerator pedals have been pressed, by checking that the displacement value of the pedals exceed the minimum threshold used to filter out unintentional inputs. For any pedal deemed to pressed, the corresponding pressed or released control signal is emitted to the rest of the system. For example, if the brake displacement value is greater than 3%, a brakePressed signal is emitted, otherwise brakeReleased will instead. The same holds true in the case of the accelerator. In the context of an automobile, braking and accelerating simultaneously is valid. However, for this system, it is assumed that neither the brakes nor accelerator will be sufficiently pressed at the same time. The control signals are used by other modules such as CruiseStateController to help simplify state transitions.

C. *ValidateSpeed*

Similar to the MoniorAccelDecel, this module also processed a specific set of inputs to produce control signals for the rest of the system. In this case, the speed of the car is monitored and a control signal representing whether the speed is within the predefined limit, is emitted. Should the car exceed 150 km/h or fall below 30 km/h, a speedInvalid signal will be emitted, otherwise speedValid is emitted instead. These signals are useful for module like CruiseStateController and CruiseSpeedManager to ensure unsafe behaviour are prevented.

D. *CruiseSpeedManager*

The configuration of the cruise speed of the cruise control system is handled by this module. It takes in the external Set, quickAccel, quickDecel signals as well as the speed and current state of the system as inputs. When the system is in a safe state, the ability to set a new cruise speed with the configuration buttons are made available. This module sued Esterel's trap to emulate switch statements with breaks.

E. *CheckCruiseSpeed*

This module takes the previous cruise speed (using Esterel's 'pre' feature) and outputs control signals that represent any invalid changes to cruise speed. It prevents the system from setting the cruise speed to unsafe or impractical values such as speeds above the speed limit.

F. *CruiseThrottleRegulator*

This module is responsible for calculating the throttle value to pass to the throttle command. Depending on the state of the cruise control system, the module will select whether the throttle command should be controlled the cruise control system or the accelerator pedal directly. If the cruise control system is handling the throttle command, then the system will calculate the throttle value using the integral control system (defined in the 'cruiseController_data.c' file) with the car speed and cruise speed values.

V. Data Handling and Interfaces

A. *CruiseController Module*

This module is the top-level Esterel module that encapsulates all the other modules under it, this module also runs the following modules in parallel: CruiseStateController, CheckCruiseSpeed, ValidateSpeed, MonitorAccelDecel, CruiseSpeedManager and CruiseThrottleRegulator. As signals in Esterel are not allowed to be explicitly shared, we solve this problem by using the signal renaming approach. All input and output signals that are exposed must be declared in this top-level module.

```
module cruiseController:
input on, off, resume, set, qAccel,
qDecel;
input brake:=0.0f: float,
speed:=0.0f: float, accel:=0.0f: float;
output cruiseState: integer,
cruiseSpeed: float, throttleCmd: float;
signal accelP, accelR, brakesP, brakesR,
speedV, speedI, csMax, csMin, init in
```

B. *MonitorAccelDecel Module*

This is a supporting module responsible for determining whether the accelerator and decelerator pedals (monitor_accel and monitor_decel) meet the 'MIN_PEDAL' threshold value to be considered as pressed, if so the output signals (accel_pressed, accel_released, brake_pressed and brake_released) are emitted, which are used in the CruiseStateController module.

```
module MonitorAccelDecel:
constant MIN_PEDAL = 3.0f : float;
input monitor_accel : float,
monitor_brake : float;
output accel_pressed, accel_released,
brake_pressed,
brake_released;
```

C. *CheckCruiseSpeed Module*

This is a supporting module that determines whether the previous cruiseSpeed from the CruiseSpeedManager(using 'pre' in Esterel) plus or minus 'INCREMENT' or 'DECREMENT' respectively is within the valid cruiseSpeed range, if not the output signals (cruiseSpeedMin, cruiseSpeedMax) are emitted, which are used in the CruiseSpeedManager.

```
module CheckCruiseSpeed:
constant MIN_SPEED = 30.0f : float;
constant MAX_SPEED = 150.0f: float;
constant INCREMENT = 2.5f: float,
DECREMENT = 2.5f: float;
input cruiseSpeed1 : float;
output cruiseSpeedMax, cruiseSpeedMin;
```

D. *ValidateSpeed Module*

This is a supporting module that determines whether the car speed is within the valid cruiseSpeed range, if not the output signals (validSpeed, invalidSpeed) are emitted, which are used in the CruiseSpeedManager.

```
module ValidateSpeed:
constant MIN_SPEED = 30.0f : float;
constant MAX_SPEED = 150.0f: float;
input validate_speed : float;
output valid_speed, invalid_speed;
```

E. *CruiseStateController Module*

This module is the main controller module that implements the state machine for the cruise control system. The module requires the inputs: on, off, resume and the internal signals from the supporting modules: acceleratorPressed, acceleratorReleased, brakesPressed, brakesReleased, speedValid and speedInvalid. The module emits the new cruiseState depending on the external and internal inputs. It also emits 'initCS' when the controller transitions from the off to the on state to be used in the CruiseSpeedManager module.

```
module CruiseStateController:
input signalOn, signalOff, signalResume,
acceleratorPressed, acceleratorReleased,
brakesPressed, brakesReleased,
speedValid,
speedInvalid;
output initCS, CruiseState: integer;
```

F. *CruiseSpeedManager Module*

The 'set1' input is from the top-level interface, the 'initCS' input is the output of the CruiseStateController module, the other inputs are from the supporting modules. The output signal 'newCruiseSpeed' is used in CruiseThrottleRegulator.

```
module CruiseSpeedManager:
constant INCREMENT = 2.5f: float,
constant DECREMENT = 2.5f: float;
constant MIN_SPEED = 30.0f : float,
constant MAX_SPEED = 150.0f: float;
input set1, quickAccel, quickDecel,
speedValid, initCS, cruiseSpeedMin,
cruiseSpeedMax;
input speed1: float;
input state: integer;
output newCruiseSpeed: float;
```

G. *CruiseThrottleRegulatorModule*

This module receives inputs 'CTRCruiseState' and 'CTRCruiseSpeed' from the CruiseStateController and CruiseSpeedManager modules respectively. The output signal 'CTRThrottleCmd' is outputted from the system and is used to control fuel flowing into the engine.

```
module CruiseThrottleRegulator:
function regulateThrottle(integer,
float, float): float;
input CTRCruiseState: integer,
CTRCruiseSpeed: float, CTRSpeed: float,
CTRAccel: float;
output CTRThrottleCmd: float;
```

VI. Testing and Verification for Causality

The cruise control system is a safety critical system, as any abnormal or incorrect behaviour could result in fatal accidents. As such, the system needs to ensure causality and must be functionally correct. Causality is defined as the relationship between cause and effect, or in this case, stimulus and response. Esterel does not ensure that a compiled program will be causal due to concurrent nature and the instantaneity of signals and the responses to them may create cyclic dependencies and consequently result in a non-causal program. Therefore, to ensure causality we took some precautions when developing the system. To prevent cyclic dependencies, we used Esterel's 'pre' feature to access the previous state or value of the signal. An example of this was when we had a variable cruiseSpeed that was being updated concurrently, to prevent the cyclic dependency we used the 'pre' feature to access the previous value of cruiseSpeed before incrementing or decrementing the value. To further ensure causality, we looked at multiple input stimuli and traced the dependencies to check for any cyclic dependencies. However, given that there are many other possible sequences, we cannot guarantee causality. We also check our system for functional correctness by using all the provided test cases and our own manual test cases to check if the output was expected given a certain input. However, these test cases don't guarantee functional correctness. In the future we could look into the ATG graphic package to further check for non-causal loops and also look into the use of the Esterel Verification Environment (XEVE) to further check for functional correctness of the system.

VII. Conclusion

In conclusion, we have established the specifications of the car cruise control system, implemented the design based on the specifications utilising Esterel with its synchronous features. The system was simulated to check for causality and functional correctness. Future work is intended to further test and verify the system using XEVE and the ATG graphic package.

Acknowledgment

The authors would like to acknowledge: 1) Associate Professor Partha Roop for providing his Lift Controller implementation and report. 2) The TAs: Keith Ren, Nathan Allen for their help in the labs. 3) The lecturer Avinash Malik

Table of Contribution

Name	**Implementation (Hours spent)**	**Report (Hours spent)**
Dylan Fu	15	4
Quentin Heng	15	4