COMPSYS 723 – ASSIGNMENT 2 REPORT

Shane du Plessis & Nikolay Petin

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

Abstract

This report covers our implementation of a cruise control system. Implemented using Esterel, we developed this system using a model-based approach. Starting with decomposing function specifications, we develop individual modules that make up the system. The result is a reactive, concurrent system that executes cruise control functionality according to the given brief.

1. Introduction

In the brief, we are tasked with creating a cruise control system. By comparing the speed of the car against the target cruise speed, the system regulates the throttle to maintain the target speed. The target speed can be set to the current speed when the system is first turned on or upon a *"set"* signal. The system must be aware of the minimum and maximum speeds, meaning that it should limit the target speed to these bounds. Furthermore, the pressing of any pedal should disable the system, and allow for the resumption of the last target speed.

In section 2, we cover the specification and provide the context digrams to describe the composition of the controller. In section 3, we explain the modules and their controlling FSMs. In section 4, we explain how we implemented this design in Esterel followed by section 5, where we talk about the validation of the design. Finally, in section 6, we conclude by reflecting on the result.

2. Specification

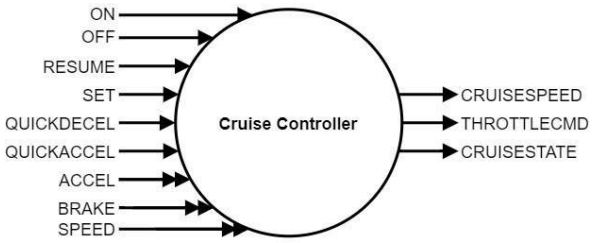


Figure 1 – First context diagram

Figure 1 shows the general context diagram for the specified controller. The input consists of control signals to turn the controller on or off, set a target speed, and resume operation. Other signals used to increment and decrement the target speed. Furthermore, the pedal signals are sent along with the current speed of the car.

The outputs consist of the target cruise speed, throttle command to control the throttle, and the current state of the controller.

This model was too simple to understand how to implement the system, so it was decomposed so that state and throttle control were two different modules.

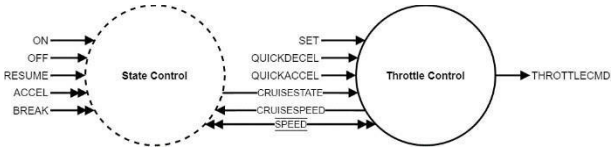


Figure 2 – Iteration of context diagram

In Figure 2, we decompose this context diagram into a state controller and throttle controller. This helps distinguish between the control and data partitions of the system. The state controller is responsible for reading status inputs and setting the cruise state signal. Throttle control is responsible for regulating the throttle according to the target speed. The target speed is also managed in this module.

We recognized that the throttle control unit could be decomposed further to help simplify implementation.

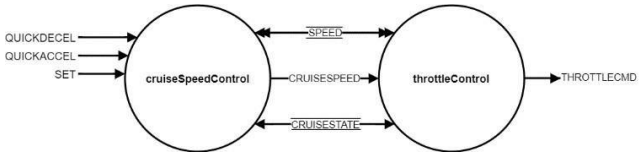


Figure 3 - Decomposition of throttle control

In Figure 3, we have decomposed the throttle control such that one module takes care of setting the target cruise speed and interpreting the cruise setting buttons and another module just for regulating the speed to match. By taking care of setting the cruise speed seperatley, the throttleControl unit is simplified, only having to worry about regulating the throttle.

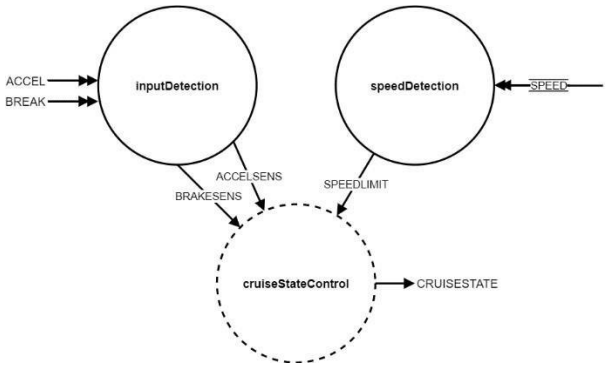


Figure 4 - Decomposition of cruiseStateControl

Lastly, we recognized that continuous inputs such as accel, brake, and speed could complicate the state controller. As seen in Figure 4, we solved this by introducing input and speed detection modules. The input detection works by comparing the pedals against predefined thresholds and emitting a discreet signal indicating that respective pedal has been pressed. Similarly, speed detection compares the current speed against a predefined threshold to emit a speed warning signal.

With appendix A, we present the final diagram showing how everything links together, with the proper variable names. It is important to note that at this point, we realized we needed a specific signal "isGoingOn" for the throttle regulating code to reset the tracking error.

3. Design

After we had decomposed the system, it was time to design the actual logic behind each module. This was achieved by drawing a finite state machine for each module. The following will go over each module.

3.1. cruiseStateControl

Figure 5 shows our cruiseStateControl module; it is responsible for determining what state the system is in. It reads the control inputs directly from the user interface (i.e. cruise control lever), the pedals, and the current speed of the vehicle to determine the current state. This state is the most important as it has to make sure not to enable cruise control over the speed limit.

3.2. cruiseSpeedControl

Appendix B shows our cruiseSpeedControl module; it is responsible for setting the current target speed. Special care has to be taken so that the target speed never exceeds the upper and lower speed limits.

3.3. throttleControl

Figure 6 shows our throttleControl module; it is responsible for regulating the throttle so that the car speed matches the target cruise speed. A simple module, only needs to know the current state of the system.

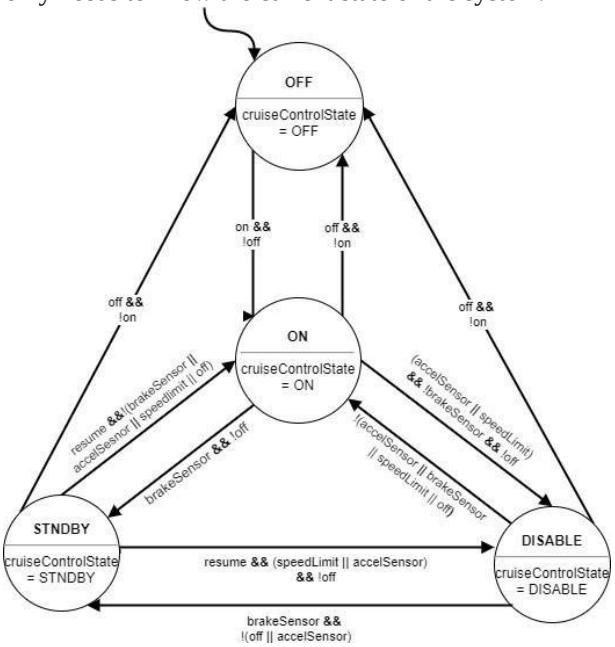


Figure 5 - cruiseStateControl FSM

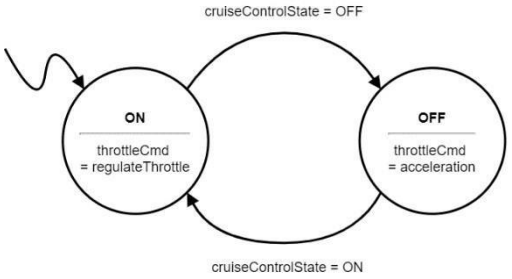


Figure 6 - throttleControl FSM

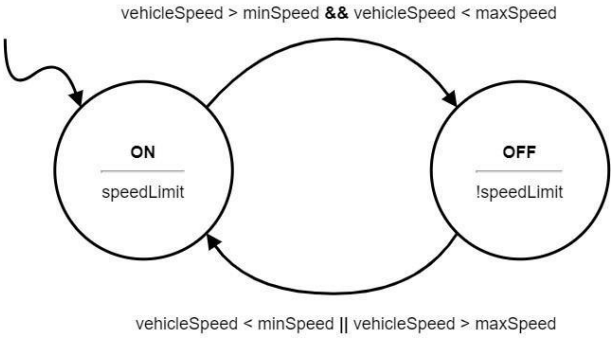


Figure 7 - speedDections FSM

3.4. speedDetection

Figure 7 shows our speedDetection module. Its only purpose is to report to cruiseStateControl if the current speed is exceeding the predefined speed limit.

3.5. inputDetection

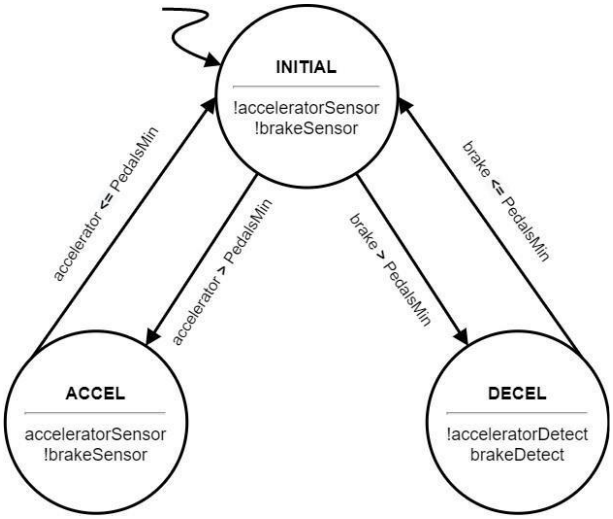


Figure 8 - inputDetection FSM

Figure 8 shows our input module. Its only purpose is to report to cruiseStateControl if any of the pedals have been pressed further than the predefined value, discreetly indicating an acceleration or braking event.

4. Implementation

The final implementation in Esterel proved to be straight forward, after some initial confusion over the syntax. By individually writing each module, the work was divided into small manageable chunks.

Despite the initial difficulty, we first started with small modules such as *throttleControl* to get comfortable with Esterel. With the help of the "Esterel Primer" document provided, we were able to develop every module. After all the modules were written, we interfaced them as seen in Figure 9. This shows all of our inputs, outputs, internal signals, and how the modules connect with them.

5. Verification

Once the system was written and interfaced, we moved on to verify its functionality. The final verification was done by using the included vector files with manual testing. The vector files provided a good starting point and were intended to automate the testing via the Esterel simulator, but we could not get them working reliability. Instead, we manually put the vector inputs into the simulator and compared the results against the given outputs, the result seemed to match as seen in appendix C. Furthermore, to make sure that our design fit to the specifications, we manually went over potential test cases that the small vector file did not cover. After this testing procedure we were confident that our system matches the requirements.

```
module cruiseControl:

%defining signals outlined as per assignment brief
input on;
input off;
input resume;
input set;
input quickAccel;
input quickDecel;
input accelerator : float;
input brake : float;
input vehicleSpeed : float;

output cruiseSpeed : float;
output throttleCmd : float;
output cruiseControlState : integer; %1 = OFF; 2 = ON; 3 = STDBY; 4 = DISABLED

signal isGoingOn : boolean in
signal speedLimit in
signal brakeSensor in
signal acceleratorSensor in

run throttleControl [
    signal isGoingOn/isGoingOn;
    signal cruiseSpeed/cruiseSpeed;
    signal vehicleSpeed/vehicleSpeed;
    signal accelerator/accelerator;
    signal cruiseControlState/cruiseControlState;
    signal throttleCmd/throttleCmd
    ]
|| run speedDetection [
    signal vehicleSpeed/vehicleSpeed;
    signal speedLimit/speedLimit
    ]
|| run inputDetection [
    signal accelerator/accelerator;
    signal brake/brake;
    signal acceleratorSensor/acceleratorSensor;
    signal brakeSensor/brakeSensor
    ]
|| run cruiseSpeedControl [
    signal set/set;
    signal quickAccel/quickAccel;
    signal quickDecel/quickDecel;
    signal vehicleSpeed/vehicleSpeed;
    signal cruiseControlState/cruiseControlState;
    signal cruiseSpeed/cruiseSpeed
    ]
|| run cruiseStateControl [
    signal on/on;
    signal off/off;
    signal resume/resume;
    signal speedLimit/speedLimit;
    signal brakeSensor/brakeSensor;
    signal acceleratorSensor/acceleratorSensor
    ]

end signal;
end signal;
end signal;
end signal;

end module
```

Figure 9 - cruiseControl interfacing

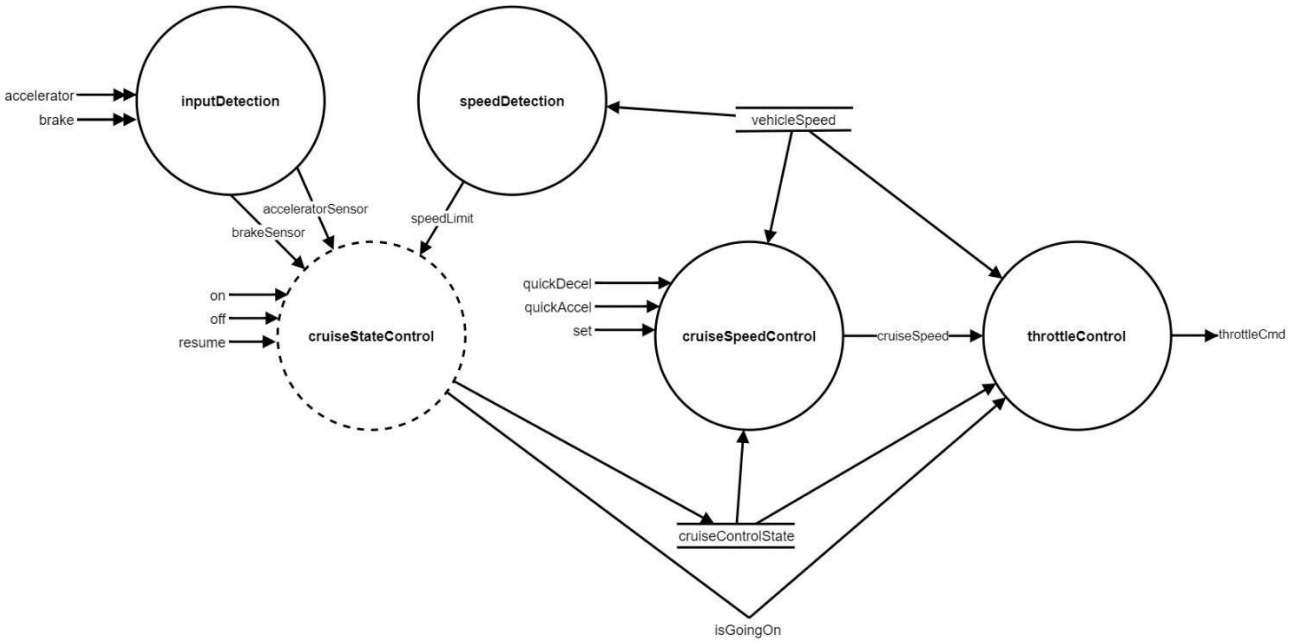
6. Conclusion

In this report, we have covered our plan, implementation, and verification of a specified cruise control system. Implemented using Esterel, we developed this system using a model-based approach. Starting with decomposing function specifications, we develop individual modules that make up the system. The result is a reactive, concurrent system that executes cruise control functionality according to the given brief.

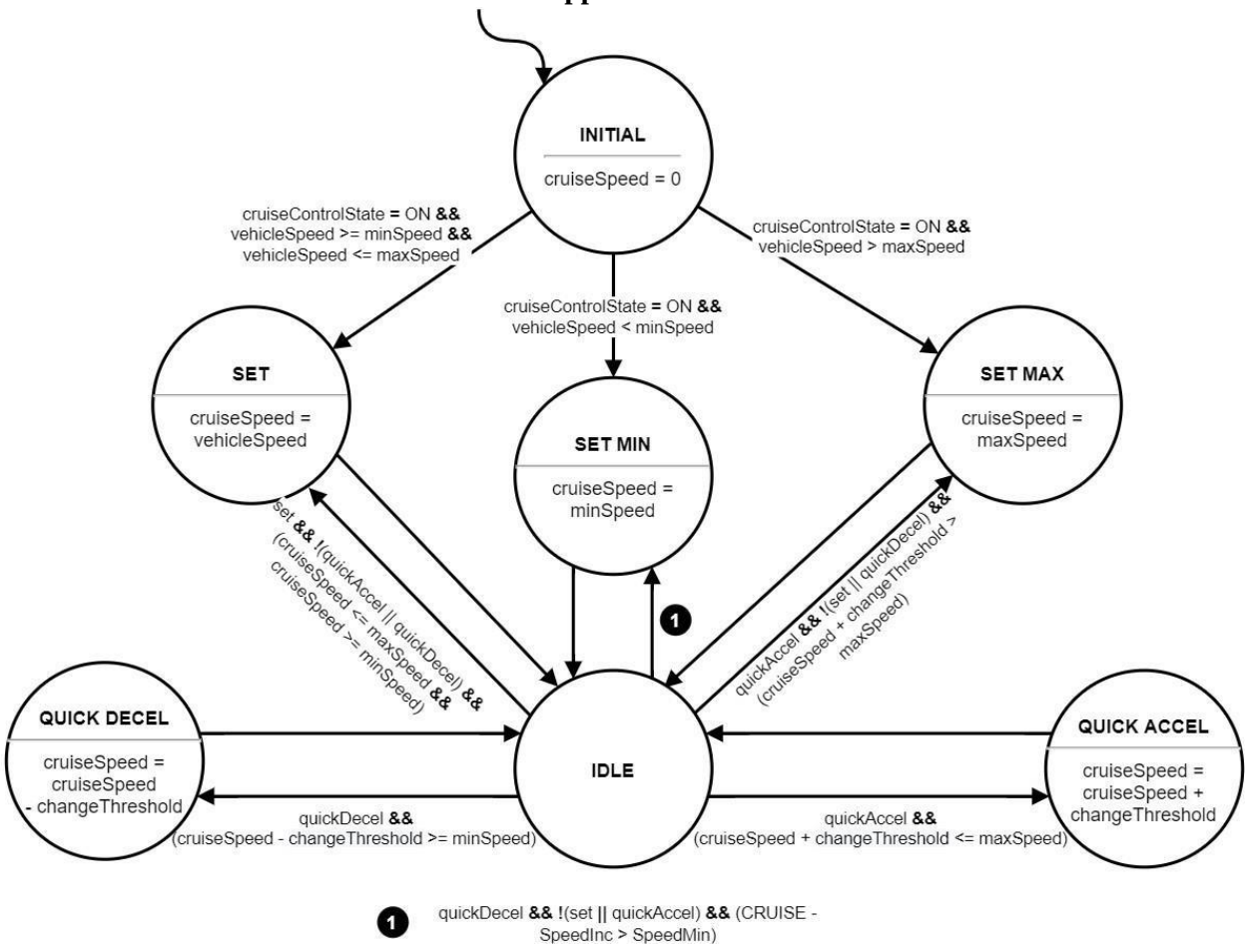
7. Time Commitment

	Shane	**Nikolay**
Tasks	**Hours Spent**	
Planning	2	7
Implementation	11	3
Testing	3	2
Documentation	1	5
Total	17	17

Appendix A



Appendix B



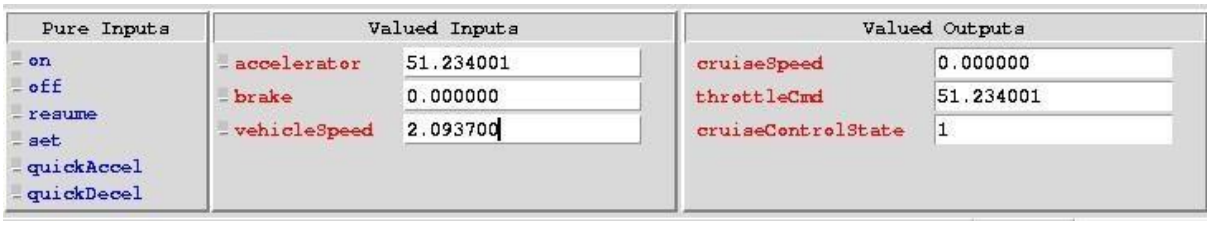
1 quickDecel && !(set || quickAccel) && (CRUISE - SpeedInc > SpeedMin)

Appendix C

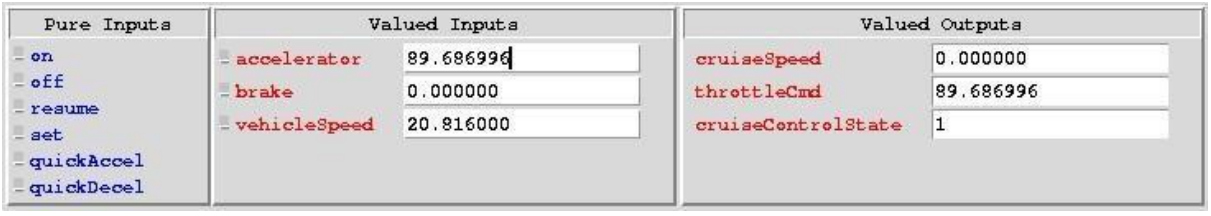
```
# On Off Resume Set QuickAccel QuickDecel Accel Brake Speed
false false false false false false 0.0 0.0 0.0
false false false false false false 0.0 0.0 0.0
false false false false false false 51.234 0.0 0.0
false false false false false false 51.234 0.0 2.0937
false false false false false false 51.234 0.0 4.1862
false false false false false false 51.234 0.0 6.2772
false false false false false false 51.234 0.0 8.3661
false false false false false false 89.687 0.0 10.453
false false false false false false 89.687 0.0 13.051
false false false false false false 89.687 0.0 15.644
false false false false false false 89.687 0.0 18.233
false false false false false false 89.687 0.0 20.816
false false false false false false 89.687 0.0 23.393
false false false false false false 89.687 0.0 25.962
false false false false false false 89.687 0.0 28.524
false false false false false false 89.687 0.0 31.078
false false false false false false 89.687 0.0 33.623
false false false false false false 0.0 0.0 36.158
true false false false false false 0.0 0.0 36.049
false false false false false false 0.0 0.0 35.94
```

```
# CruiseSpeed ThrottleCmd CruiseState(OFF-1 ON-2 STDBY-3 DIS-4)
0.000000 0.000000 1
0.000000 0.000000 1
0.000000 51.234001 1
0.000000 51.234001 1
0.000000 51.234001 1
0.000000 51.234001 1
0.000000 51.234001 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 89.686996 1
0.000000 0.000000 1
36.049000 0.000000 2
36.049000 0.938827 2
```

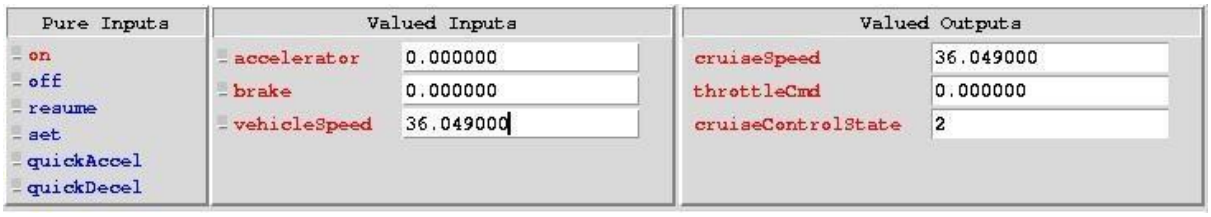
vectors.in vs vectors.out



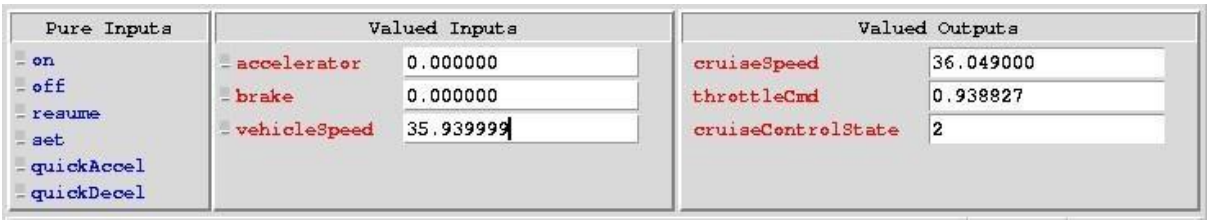
Line 5



Line 13



Line 20



Line 21