723 Report Group 31

Tai Martinez Grant, Iain Ross, Fangping Wang

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

Abstract

This report discusses the main components of a simple cruise control system. This includes each design decision made when designing this system, each intermediate step taken to reach the final product, and the intended purpose of each identified component.

1. Introduction

A cruise control system is a system which is used to regulate and control the speed of a moving vehicle. The purpose of this specific system is to regulate the speed of a car when the driver specifically desires and when certain criteria are met. These criteria are based on the different states the cruise control will be operating.

For our assignment, we created this system using Esterel. Esterel is a synchronous programming language, primarily used for control-dominated software or hardware reactive systems. Esterel is advantageous due to its deterministic behaviour and precise timing. It allows programs to react in well defined ways, with no race conditions making it ideal for real time systems.

2. Main elements of the solution

The requirements for this solution were specifically the inputs and outputs of the system, and the logical and functional behaviour in the logic. This can be seen in the diagrams we created for our design, and the code snippets provided in the screenshots below.

Our solution required components designed both in the Esterel programming language and in C. Within our Esterel code we had 2 modules - one essentially containing our finite state machine, and the other acting as our top level module. Our C file contained the code which implemented the controller algorithm, as well as the user defined type. It also contains the enumeration for the different states we are using for the cruise control (OFF,ON,STBY,DISABLE).

2.1. C Program file

Because Esterel is unable to handle user defined types, this was done in C. As well, we were provided the necessary code to implement the control algorithm in C.

2.2. Esterel Program file

Our esterel program contains two modules as previously mentioned.

Top Level Module

This module was created to implement the Unit Control module, and this was done so that it would be able to manage other modules as well and execute them concurrently.

Unit control Module

This module essentially contains a practical implementation of the FSM designed in the diagrams below. This FSM consists of 4 total states, each corresponding to its own cruise state.

OFF
Within this state, the cruise control is off, and the driver controls the speed of the car manually. The only way to exit this state is for the driver to provide an input in the form of a button press.

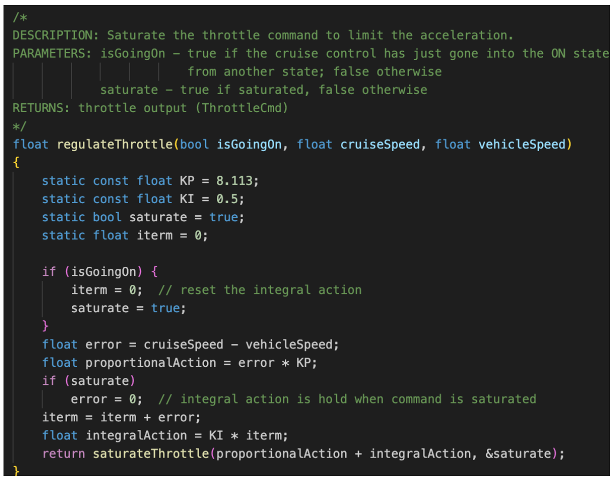
ON
Within this state the cruise control is on and controls the speed of the car, adjusting when necessary.

STDBY
The standby state ensures that the cruise control functionalities are temporarily paused, and will resume when conditions allow and the driver wishes.

DISABLE
In this state the cruise control has been forcibly turned off based on current conditions and will return to regular behaviour once conditions have settled.

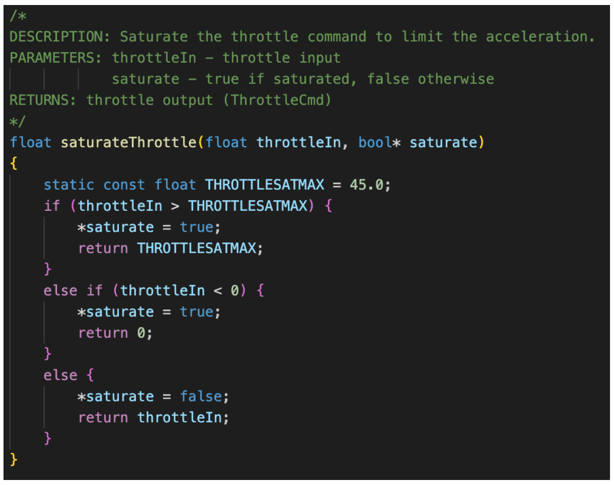
2.3. Controller Algorithm

The throttle regulation was implemented using a PI controller. This was provided to us and can be seen in the figure below.



```
/*
DESCRIPTION: Saturate the throttle command to limit the acceleration.
PARAMETERS: isGoingOn - true if the cruise control has just gone into the ON state
                        from another state; false otherwise
            saturate - true if saturated, false otherwise
RETURNS: throttle output (ThrottleCmd)
*/
float regulateThrottle(bool isGoingOn, float cruiseSpeed, float vehicleSpeed)
{
    static const float KP = 8.113;
    static const float KI = 0.5;
    static bool saturate = true;
    static float iterm = 0;

    if (isGoingOn) {
        iterm = 0;  // reset the integral action
        saturate = true;
    }
    float error = cruiseSpeed - vehicleSpeed;
    float proportionalAction = error * KP;
    if (saturate)
        error = 0;  // integral action is hold when command is saturated
    iterm = iterm + error;
    float integralAction = KI * iterm;
    return saturateThrottle(proportionalAction + integralAction, &saturate);
}
```



```
/*
DESCRIPTION: Saturate the throttle command to limit the acceleration.
PARAMETERS: throttleIn - throttle input
            saturate - true if saturated, false otherwise
RETURNS: throttle output (ThrottleCmd)
*/
float saturateThrottle(float throttleIn, bool* saturate)
{
    static const float THROTTLESATMAX = 45.0;
    if (throttleIn > THROTTLESATMAX) {
        *saturate = true;
        return THROTTLESATMAX;
    }
    else if (throttleIn < 0) {
        *saturate = true;
        return 0;
    }
    else {
        *saturate = false;
        return throttleIn;
    }
}
```

2.4. System Design Diagrams

2.4.1. Context Diagram

The context diagram was drawn to understand the environment of our system. It clearly lays out all the inputs and outputs and gives us an idea of how they might fit in together. The context diagram is shown in Figure 1.

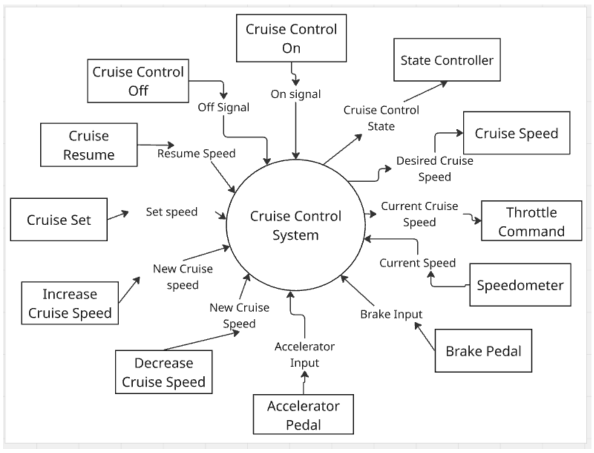


Figure 1: Context Diagram

2.4.2. First Level Refinement Diagram

The First Level Refinement Diagram was made to give us an understanding of the different modules that make up our system. After analysing the Context diagram, we finalised our modules as the "Cruise State Manager", "Speed Regulation Unit" and the "Throttle Saturation Logic". This diagram is shown in Figure 2

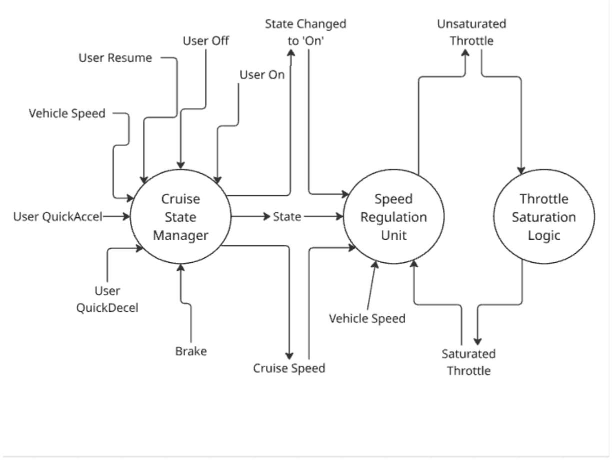


Figure 2: First Level Refinement Diagram

2.4.3 Cruise State Manager FSM Diagram

To finalise the implementation of our Cruise State Manager, we created an FSM diagram to precisely define the behavior of this module. This is shown in Figure 3. The Cruise State Manager has four states; OFF, ON, DISABLE, and STDBY (Standby). The states can change depending on the inputs into the system. The system starts in the OFF state. The user can change to the OFF state from any state by pressing the "Off" button. From the OFF state, if the "On" button is pressed, the system goes to the ON state. In the ON state, the system checks for input on both the brake pedal and accelerator pedal. If the accelerator pedal is pressed, the system changes to the DISABLE state. If the Brake pedal is pressed, the system changes to the STDBY state. When in the DISABLE state, the system checks if the accelerator pedal is no longer pressed and if the current speed is in the valid range of the system before moving back to the ON state. From the STDBY state the system can either move into the DISABLE or ON state. In the STDBY state the system is checking for the "Resume" Button to be pressed. if the "Resume button is pressed and the current speed is valid, the system returns to the ON state. However, it goes to the DISABLE state if the "Resume" Button is pressed and if either the accelerator pedal is pressed or if the speed is not valid. This functionality is shown in Figure 3.

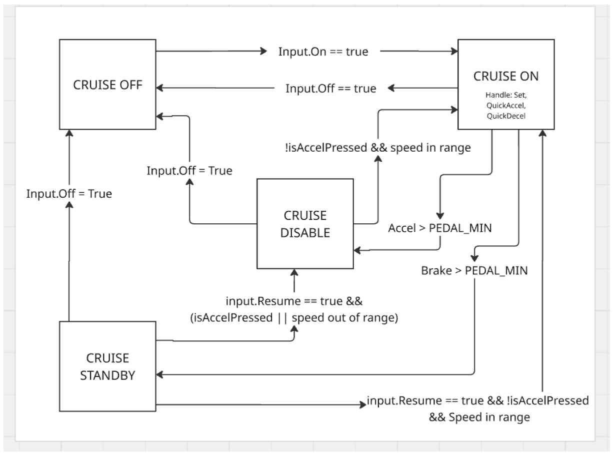

CRUISE OFF
Input.On == true
CRUISE ON
Handle: Set, QuickAccel, QuickDecel
Input.Off == true
!isAccelPressed && speed in range
Input.Off = True
Input.Off = True
CRUISE DISABLE
Accel > PEDAL_MIN
Brake > PEDAL_MIN
input.Resume == true && (isAccelPressed || speed out of range)
CRUISE STANDBY
input.Resume == true && !isAccelPressed && Speed in range

Figure 3: System State Diagram

2.4.4 *Speed Manager System Diagram*

The Cruise Speed Manager is the system designed to handle the logic behind the current cruise control speed. The outputs of this system are controlled by the current state and button inputs from the user. When the Cruise Control system is off, the Cruise output speed is always zero. When the cruise control is turned on, the speed is set through three buttons. "Set", "QuickAccel", and "QuickDecel". When "Set" is pressed, the current cruise speed is set to the vehicle's current speed. When either "QuickAccel"/"QuickDecel" is pressed, the current cruise speed is set to the vehicle's current speed +/- SpeedInc. This logic is shown in Figure 4.

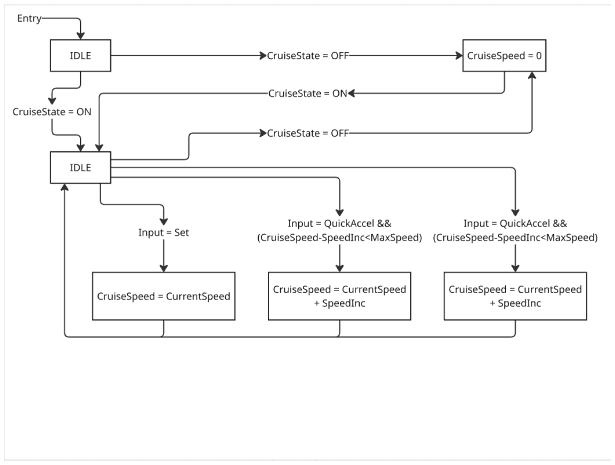

Entry
IDLE
CruiseState = OFF
CruiseSpeed = 0
CruiseState = ON
CruiseState = ON
CruiseState = OFF
IDLE
Input = Set
Input = QuickAccel && (CruiseSpeed-SpeedInc<MaxSpeed)
Input = QuickAccel && (CruiseSpeed-SpeedInc<MaxSpeed)
CruiseSpeed = CurrentSpeed
CruiseSpeed = CurrentSpeed + SpeedInc
CruiseSpeed = CurrentSpeed + SpeedInc

Figure 4: Speed Controller State Diagram

2.4.5 *Throttle Manager System Diagram*

The Throttle Manager is a system in place to prevent the vehicle speed from increasing or decreasing too rapidly. This system is used anytime that the vehicle cruise speed changes.

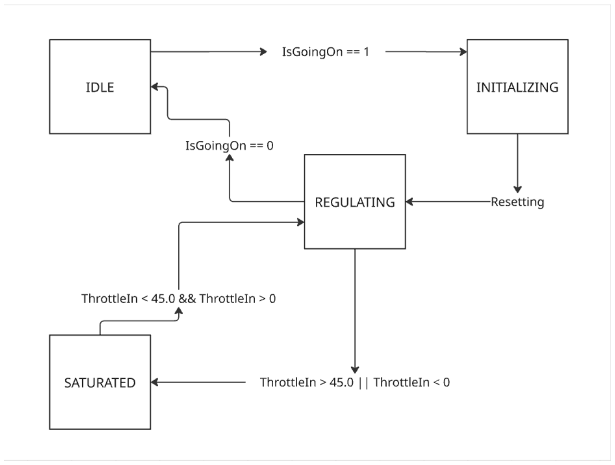
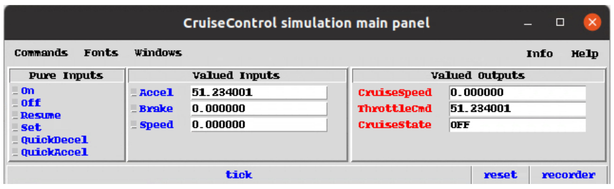


Figure 5: Throttle Controller State Diagram

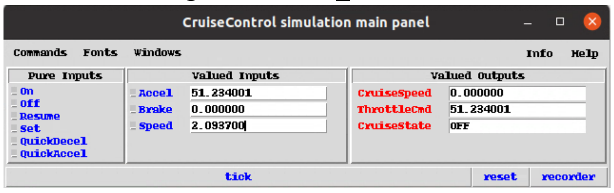
3. Results

We tested the final solution using combinations of input vectors and comparing the output to the expected solutions. The "vector_in" and "vector_out files used for testing are provided with the solution. Below are results of tests done with different inputs from the mentioned files. We can verify that these results match with the expected outputs.



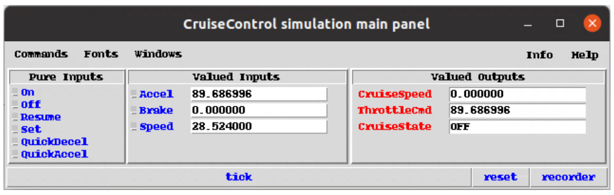
Accel 51.234001
Brake 0.000000
Speed 0.000000
CruiseSpeed 0.000000
ThrottleCmd 51.234001
CruiseState OFF

Figure 6: Vector_In test 3



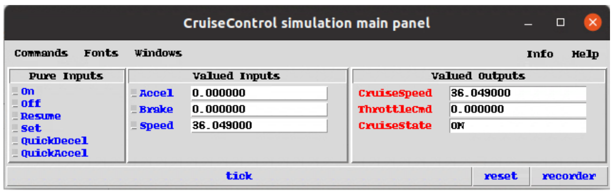
Accel 51.234001
Brake 0.000000
Speed 2.093700
CruiseSpeed 0.000000
ThrottleCmd 51.234001
CruiseState OFF

Figure 7: Vector_In test 4



Accel 89.686996
Brake 0.000000
Speed 28.524000
CruiseSpeed 0.000000
ThrottleCmd 89.686996
CruiseState OFF

Figure 8: Vector_In test 15



Accel 0.000000
Brake 0.000000
Speed 36.049000
CruiseSpeed 36.049000
ThrottleCmd 0.000000
CruiseState ON

Figure 9: Vector_In test 19

Conclusions

Overall, the implementation of the project taught us a lot about the decision making behind system design, and a lot about how esterel code is to be applied.

Acknowledgements

Our team would like to thank our lecturer Avinash Malik, and the teaching assistants who were able to support us in the design and implementation of our project.

4. References

[1] Ogata, K., “Modern control engineering”, 2020