Embedded Control Systems 5LIJ0
Project 3: : Controller Design over Distributed Architectures

Idan Grady (1912976)

April 15, 2023

1 Introduction

The goal of this project is to create a controller for a dynamic system that meets a predefined set of requirements while taking the limits of a distributed embedded platform into mind. We shall concentrate on the architecture of a Lane Keeping Assist System (LKAS) platform.

LKAS is a vehicle safety system that assists in keeping the vehicle inside its allotted lane, notifying the driver of potentially hazardous circumstances, and activating emergency brakes when a collision is imminent. The technology detects the lane and surrounding objects using the front camera, while the gyroscope and speed sensor calculate the steering angle required for lane-keeping. The LKAS can also work autonomously to keep the car within the lanes.

2 Common

2.0.1 Requirement

Order	Name	Evaluate	Show	Reference
1	Response Time SteeringOutout <= 10 ms	☒	☒	
2	Event Chain "Frame_image" Latency step first to last <= 150 ms	☒	☒	
3	Event Chain "Steer_Steeroutput" Latency step first to last <= 300 ms	☒	☒	
4	Event Chain "Gyro_steer_output" Latency step first to last <= 300 ms	☒	☒	
5	Event Chain "Frame_SteeringOut1" Latency step first to last <= 300 ms	☒	☒	
6	Event Chain "Frame_SteeringOut" Latency step first to last <= 300 ms	☒	☒	
7	Latency from "start SteeringOutout on P5" to "start SteeringOutout on P5" <= 170 ms	☒	☒	
8	Latency from "start Warning on P4" to "start Warning on P4" <= 170 ms	☒	☒	
9	Latency from "start DashBoard on P2" to "start DashBoard on P2" <= 150 ms	☒	☒	
10	Event Chain "Steering_Frequency" Latency step first to last < 170 ms	☒	☒	
11	Event Chain "WarningUpdate" Latency step first to last < 170 ms	☒	☒	
12	Event Chain "Front_Dash" Latency step first to last <= 300 ms	☒	☒	
13	Event Chain "On_Output" Latency step first to last < 80 ms	☒	☒	

Figure 1: requirement

2.0.2 Simulation parameters

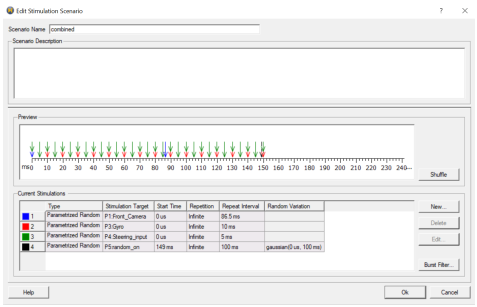


Figure 2: Caption

Before comparing the two cases, it is critical to understand the simulation parameters present in both designs of the LKAS. These simulation parameters, which include start time, signal, repetition, repeat interval, and random variation, are critical in ensuring that the LKAS system operates efficiently and safely. Before we

3 Design

3.1 Case 1

The task in Case I is to design a platform using electronic control units (ECUs) and CAN buses, with a budget of 850 euros. To achieve it, I have used the following components.

Component	Quantity	Type	Cost per Unit (in euros)	Total Cost (in euros)
Electronic Control Units	4	Generic TDMA	30	120
Electronic Control Units	1	Generic OSEK	50	50
CAN Buses	1	45 kbits/sec	200	200
CAN Buses	1	65 kbits/sec	350	350
Total Cost (in euros)	-	-	-	720

Table 1: Component Parameters and Costs CASE 1

3.1.1

The following graph depicts the task distribution among the electronic control units (ECUs) in the Case I platform. The platform is made up of four generic TDMA-based ECUs, one generic OSEK ECU, and two different bit rate CAN buses. Each task is assigned to an ECU based on the computing power and communication requirements.

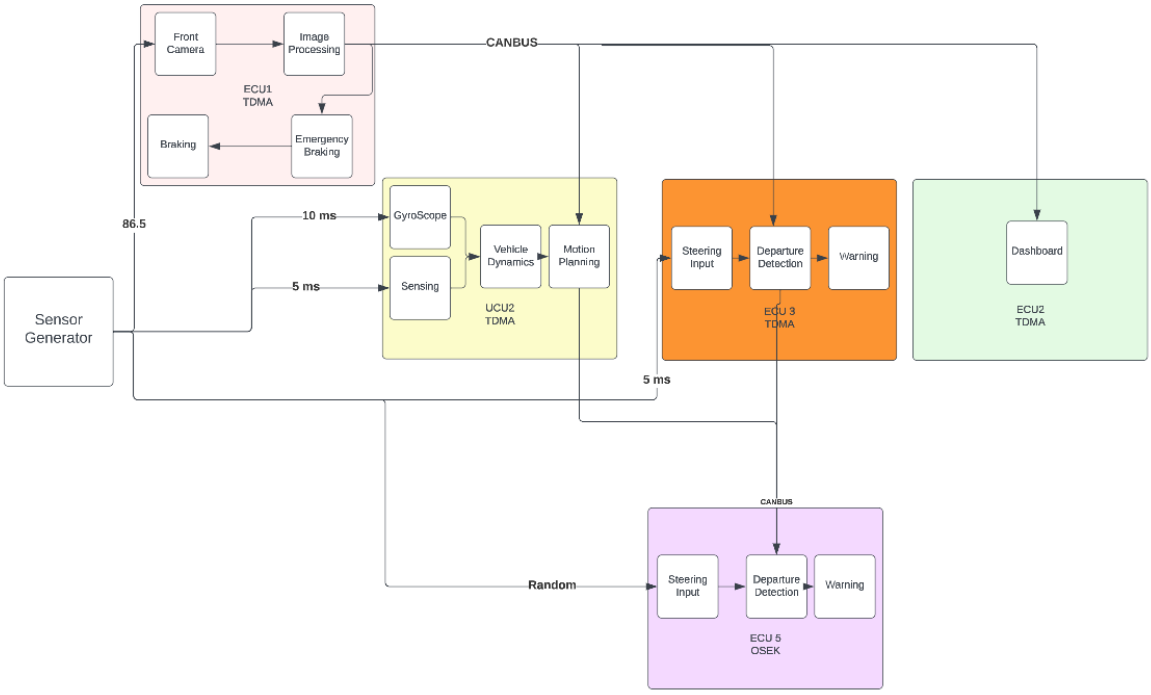


Figure 3: CASE 1 Distibuted Control

While there are several ways to distribute tasks among the ECUs, the approach used in this design for Case I was thought to be the most time-efficient. Despite the additional manual effort, this method

provided more control and a better understanding of the software. The design ensures that each ECU has enough computing power and communication bandwidth to handle its assigned tasks by manually allocating tasks to specific ECUs.

3.1.2

While there are several ways to distribute tasks among the ECUs, the approach used in this design for Case I was thought to be the most time-efficient. Despite the additional manual effort, this method provided more control and a better understanding of the software. The design ensures that each ECU has enough computing power and communication bandwidth to handle its assigned tasks by manually allocating tasks to specific ECUs.For Departure Detection, the operator is also used between Vehicle Dynamics and Steering Input tasks, as well as between Motion Planning, Departure Detection, and the on/off button for Steering Assistance. The AND operator helps to prevent errors and ensure the proper operation of the system by ensuring that each task has access to all necessary data before beginning its execution.

3.1.3

Before analyzing the actual results and determining whether the design meets the Case I requirements, lets examine the simulator simulation. The simulation provides a visual representation of the system's behavior and can assist in identifying any potential issues or areas for improvement. By watching the simulation, you can see how the various tasks interact with one another and how data flows through the system.

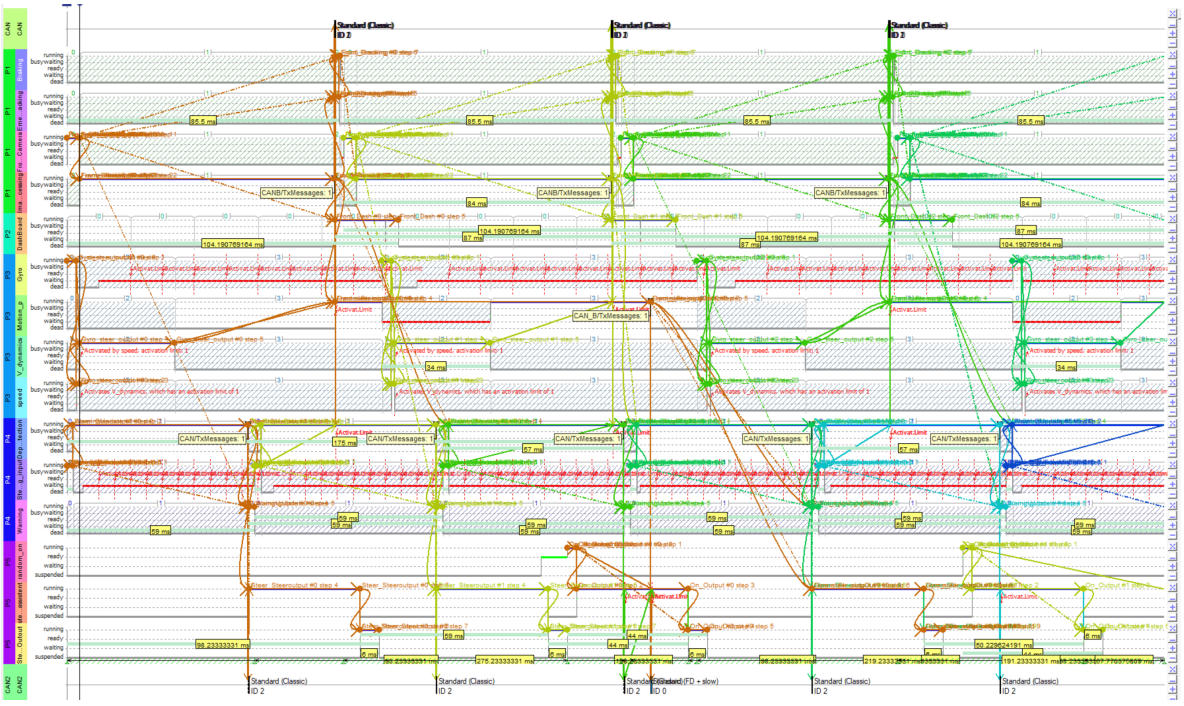


Figure 4: Simulation CASE 1

The simulation results for Case I show that the design does not meet the requirements in the first cycle. However,from a first glance, it looks like the system's performance improves in subsequent cycles as tasks are distributed more efficiently among the ECUs.

3.1.4

Order	Name	Failed	Successful	Critical	Evaluate	Show	Reference
1	Response Time SteeringOutout <= 10 ms	0% (0)	100% (1009)	-	☒	☒	
2	Event Chain "Frame_image" Latency step first to last <= 150 ms	1.5% (9)	98.5% (598)	-	☒	☒	
3	Event Chain "Steer_Steeroutput" Latency step first to last <= 300 ms	0% (0)	100% (909)	-	☒	☒	
4	Event Chain "Gyro_steer_output" Latency step first to last <= 300 ms	0% (0)	100% (542)	-	☒	☒	
5	Event Chain "Frame_SteeringOut1" Latency step first to last <= 300 ms	10.7% (65)	89.3% (542)	-	☒	☒	
6	Event Chain "Frame_SteeringOut" Latency step first to last <= 300 ms	0.8% (5)	99.2% (602)	-	☒	☒	
7	Latency from "start SteeringOutout on P5" to "start SteeringOutout on P5" <= ...	0% (0)	100% (1008)	-	☒	☒	
8	Latency from "start Warning on P4" to "start Warning on P4" <= 170 ms	0% (0)	100% (907)	-	☒	☒	
9	Latency from "start DashBoard on P2" to "start DashBoard on P2" <= 150 ms	1.5% (9)	98.5% (596)	-	☒	☒	
10	Event Chain "Steering_Frequency" Latency step first to last < 170 ms	0% (0)	100% (909)	-	☒	☒	
11	Event Chain "WarningUpdate" Latency step first to last < 170 ms	0% (0)	100% (909)	-	☒	☒	
12	Event Chain "Front_Dash" Latency step first to last <= 300 ms	0% (0)	100% (607)	-	☒	☒	
13	Event Chain "On_Output" Latency step first to last < 80 ms	6.5% (30)	93.5% (430)	-	☒	☒	

Figure 5: Constraints

After reviewing the simulation results for Case I, it is clear that the design does not fully meet the task requirements. There are four specific requirements that the system does not meet. These include the image latency requirement of less than 150ms, the steering output latency requirement of less than or equal to 300ms, and the time required between the execution of the random button and the activation of the steering assistance. Despite these failures, the system achieves a successful execution rate of more than 90% for all tasks. The majority of the failures are related to latency requirements, which can be difficult to meet in real-time systems. Although I had budget and perhaps could have found a solution. With relation to time managment I decided to leave it as it is.

3.2 Case 2

3.2.1

Component	Quantity	Type	Cost per Unit (in euros)	Total Cost (in euros)
Electronic Control Units	4	Generic TDMA	30	120
Electronic Control Units	1	Generic OSEK	50	50
FlexRay Bus	1	5 Mbits/sec	450	450
Total Cost (in euros)	-	-	-	620

We used the same number of ECUs as in Case I, but this time with one FlexRay bus instead of two CAN buses, as specified in the task. The FlexRay parameters were left to the user's discretion, and the selected bus had a speed of 5 Mbits/second and a cost of 450 euros. The total cost of this platform is 620 euros, which is well within the 850 euro limit.

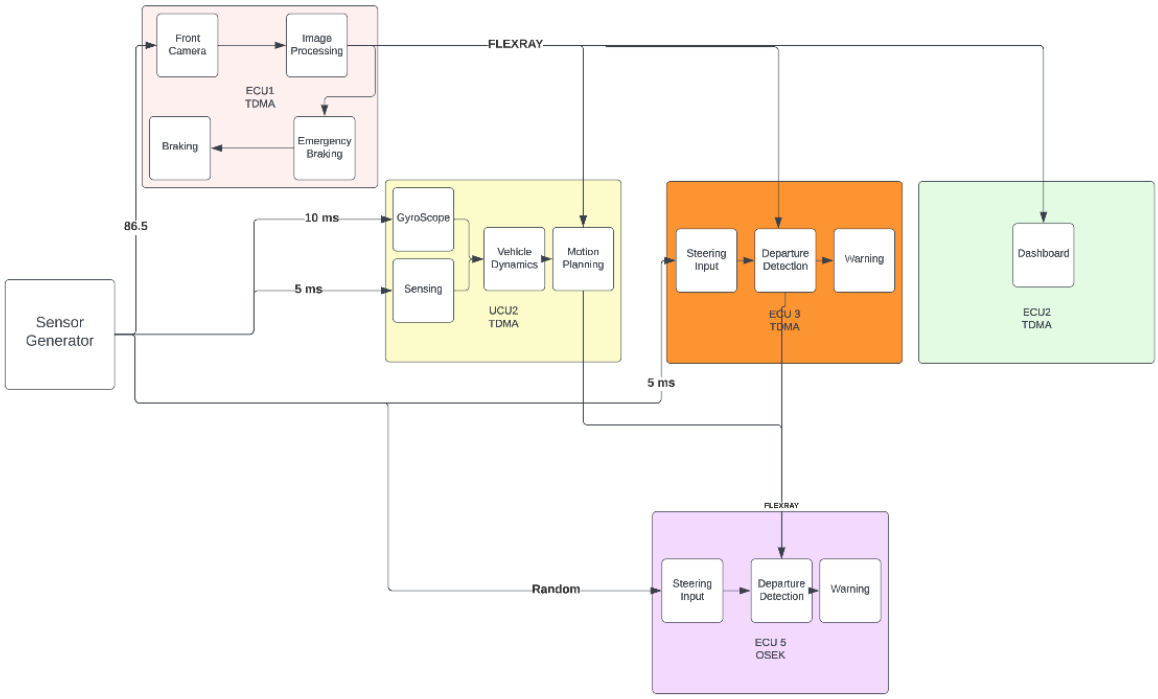


Figure 6: CASE 2 ECUs

There is no significant difference in the distribution of ECUs in Case II compared to Case I because most processors still use TDMA. The only difference is that instead of two CAN buses, we now have a single FlexRay bus with a unique set of parameters that must be defined or selected.

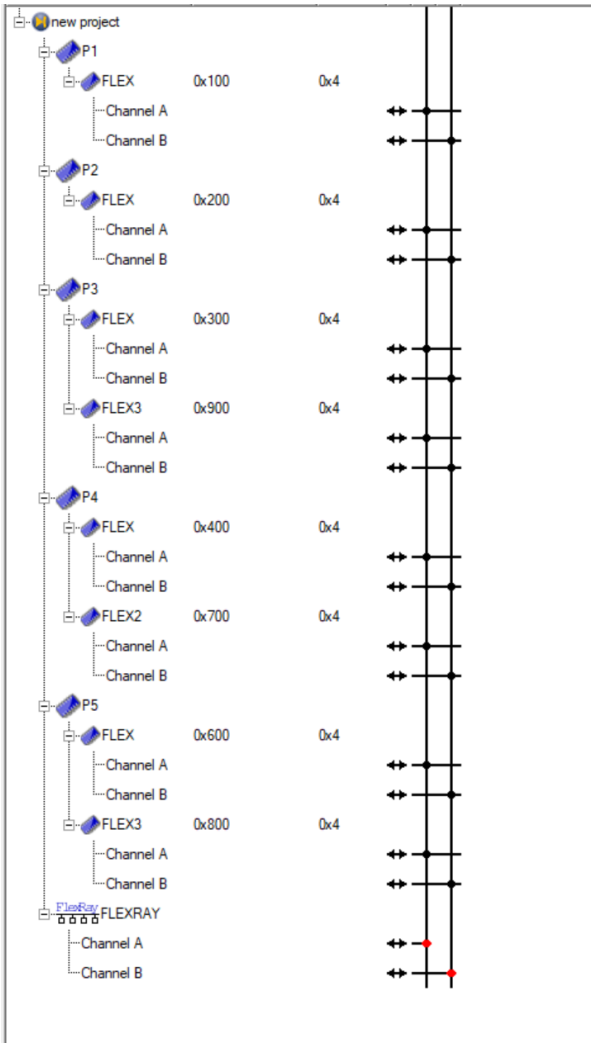


Figure 7: CASE 2 ECUs

There are some differences between CAN and FLEX buses in my implementation. I had to distribute the bus differently because we only had one. I employed three distinct sync channels: A, B, and C. Image processing communicated with Vehicle Dynamics, Motion Planning, and Departure Detection via Channel A. Channel B was used to communicate with Motion Planning, while Channel C was used to communicate with Departure Detection and Steering Assist. This distribution was made intuitively, and it may not be the best way to do it, or even the correct way. Because Motion Planning and Departure Detection share different messages, a separate sync as well as message (A channel) should be established. The figure above depicts this distribution. It's worth noting that in my implementation, a letter was treated as four bytes. In addition, I sent 8 bytes instead of 6 between the ECUs for departure detection and steering assistance. This may have had an effect on overall performance and should be considered when analyzing the results.

3.2.2

Lets check simulation

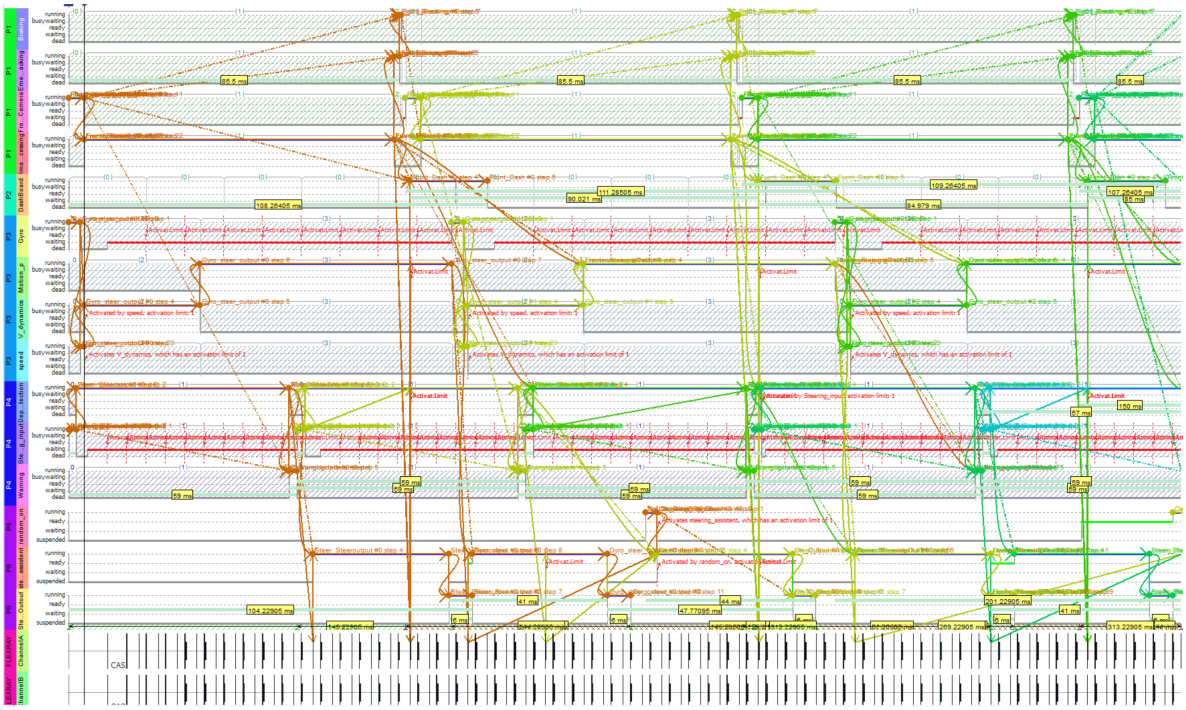


Figure 8: Simulation CASE 2

When we examine the simulation for CASE 2, we notice a similar pattern to that of CASE 1, in which the first cycle of execution fails to meet the requirements, but subsequent cycles improve.

3.2.3

1	✓ Response Time SteeringOutout <= 10 ms	0% (0)	100% (419)	-	✗	✗
2	! Event Chain "Frame_image" Latency step first to last <= 150 ms	1.2% (3)	98.8% (255)	-	✗	✗
3	✓ Event Chain "Steer_Steeroutput" Latency step first to last <= 300 ms	0% (0)	100% (384)	-	✗	✗
4	✓ Event Chain "Gyro_steer_output" Latency step first to last <= 300 ms	0% (0)	100% (229)	-	✗	✗
5	! Event Chain "Frame_SteeringOut1" Latency step first to last <= 300 ms	15.5% (40)	84.5% (218)	-	✗	✗
6	! Event Chain "Frame_SteeringOut" Latency step first to last <= 300 ms	1.2% (3)	98.8% (255)	-	✗	✗
7	✓ Latency from "start SteeringOutout on P5" to "start SteeringOutout on P5" <= ...	0% (0)	100% (419)	-	✗	✗
8	✓ Latency from "start Warning on P4" to "start Warning on P4" <= 170 ms	0% (0)	100% (382)	-	✗	✗
9	! Latency from "start DashBoard on P2" to "start DashBoard on P2" <= 150 ms	1.2% (3)	98.8% (253)	-	✗	✗
10	✓ Event Chain "Steering_Frequency" Latency step first to last < 170 ms	0% (0)	100% (384)	-	✗	✗
11	✓ Event Chain "WarningUpdate" Latency step first to last < 170 ms	0% (0)	100% (384)	-	✗	✗
12	✓ Event Chain "Front_Dash" Latency step first to last <= 300 ms	0% (0)	100% (258)	-	✗	✗
13	! Event Chain "On_Output" Latency step first to last < 80 ms	4.3% (8)	95.7% (177)	-	✗	✗

Figure 9: Constraint CASE 2

It seems like a similar pattern between the two CASEs in turms of meeting the requirements.

3.3 Comparison

To compare the two Cases , we would observe the differences in the failed tasks. In general, we can see that Case 2 had slightly higher success rates in meeting the requirements than Case 1. Case 2 demonstrated higher success rates for latency requirements in particular. However, the success rates for the steering output latency requirements in Case 2 were slightly lower. Despite this, both cases met the majority of the requirements with high success rates.

Requirement	Case 1 Success Rate	Case 2 Success Rate
Event chain "on Output" latency <= 80 ms	93.4	95.7
Latency from start Dashboard to Dashboard <=150 ms	98.5	98.8
Event chain Frame - image <=150 ms	98.5	98.8
Event chain Frame - steering output1 <=300 ms	89.3	84.5
Event chain Frame - steering output2 <= 300 ms	99.2	98.2

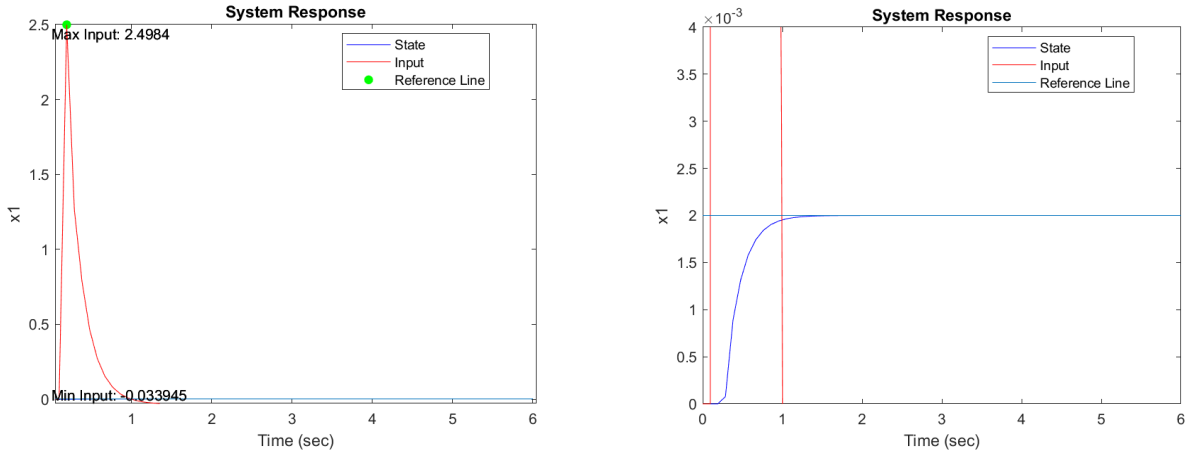
4 Matlab: CASE1 and CASE2

The delay was calculated in the implementation using the first ECUs cycle with image processing tasks, hence, 86.5 ms. This is due to the fact that following actions in each portion of the system will need to wait for the image processing to complete before proceeding (using the END and CAN/FlexRay operators). To achieve the value of h, we simply increased the resulting delay of 0.0952 by a scaling factor of 1.1. This strategy helps us to ensure that the system functions within the required latency and temporal limits while accounting for the image processing task's delay.

4.0.1 Note

Because I largely used TDMA in my approach, I didn't have to adjust my actual scheduling or ECU distribution. As a result, the latency and h calculated using the initial ECUs with image processing duties should be applied to both situations.

4.0.2 Result



(a) Close-up of input signal

(b) Response and convergence time

Figure 10: Double figure with two subplots

As seen in the figures, the system converges to the reference point in less than one second. Furthermore, the maximum input is 2.498, which is less than the 12V battery constraint. This would assure that the battery's life will be extended.

MatlabCode

```
delay = 86.5/1000;
h =delay*1.1;
numStates =2000;

%DEfine A
A = [-520 -220 0 0 0 ;
     220 -500 -999994 0 2*10.^8;
     0 1 0 0 0;
     0 0 66667 -0.1667 -1.3333*10.^7;
     0 0 0 1 0
];

B = [1000 ; 0 ;0 ;0; 0];
C = [0 0 0 0 1 ];

%Prepare for augment
A_disc  = expm(A*h);
sys_cont = ss(A, B, C, 0);

sys_disc = c2d(sys_cont, h); % the output can be changed
A_disc = sys_disc.a;
B_disc =sys_disc.b;

inv_A = inv(A_disc);

r_1_0 = (h - delay)*B;
r_1_t = delay*B;


%augment the matrix including both old input and state.
A_augment = [A_disc r_1_t ;zeros(1,6)]; % should be (4+1)x (4+1)
B_augment = [r_1_0;1];  % should be (4+1, 1)
C_augment = [C 0];

% build Q matrix. should panalize state 3 and 4
Q = eye(6);
Q(3,3) =4^13;
Q(4,4) =0.04;
Q(1,1) =140;
Q(1,2) =130;
Q(1,5) =250;
Q(2,2) =70;
Q(4,2) =4.1;
Q(4,2) =-10;
Q(3,2) =-10;
Q(5,2) =30;
Q(5,5) =40;
Q(5,3) =30;
Q(5,1) =30;


%compute LQR
[X, L, G] = dare(A_augment,B_augment, Q, 0.5);
```

```
gamma = [B_augment A_augment*B_augment A_augment^2*B_augment A_augment
    ^3*B_augment A_augment^4*B_augment A_augment^5*B_augment];
if (det(gamma)== 0)
disp('unControllable')
else
disp('Controllable')
end

%place poles
%K = -acker(A_augment,B_augment,alphas);
K =-G;
% forward gain
F = 1/(C_augment*inv(eye(6)-A_augment-B_augment*K)*B_augment);

r = 0.002; % reference

x_ = [0; 0 ;0 ;0 ;0;0 ];
tol = 1e-3; % set a tolerance for convergence
error = Inf; % initialize error to a large value
t_converge = 0; % initialize the time at which convergence is achieved


output_x = zeros(numStates, length(x_)); % to store values
input(2) = 0; input(1)=0;
time(2) = h; time(1) =0; % time initial conditions
error = 0;
length_itter = length(x_)/h;
condition_to_stop =10;
for i=2:length_itter

    y(i) = C_augment*x_;
    u  =K*[x_] +F*r;

    x_ = A_augment*x_ + B_augment*u;
    output_x(i+1,:) = x_';

    %check constraints
    if(checkConditions(x_, u,[3,4], 35, 12))
       disp('Conditioned falied');
    end

    %Stores the input and time
    input(i+1) = u;
    time(i+1) = time(i) + h;

    % compute error and check for convergence
    error = abs(output_x(i+1, 1) - r);
    if error < tol && t_converge == 0 % check if converged
        t_converge = time(i+1); % record the time of convergence
    end

    % break the loop if converged
    if t_converge > 0
        condition_to_stop =condition_to_stop-1;
        if(condition_to_stop==0)
```

```
        break
        end
    end
end


max_input = max(input);
min_input = min(input);

plot(time(1:i), y, 'b');
hold on;
plot(time(1:i+1), input, 'r')
hline = refline([0 r]);

% Add text to the figure to show the maximum and minimum input values
text(time(1), max_input, ['Max Input: ' num2str(max_input)], '
   HorizontalAlignment', 'left', 'VerticalAlignment', 'top')
text(time(1), min_input, ['Min Input: ' num2str(min_input)], '
   HorizontalAlignment', 'left', 'VerticalAlignment', 'bottom')

legend('State', 'Input', 'Reference Line', 'Location', 'best')
title('System Response');
xlabel('Time (sec)');
ylabel('x1');
ylim([0, 2*r]); % set y-axis limit
```