Eduardo Vargas Gonzalez – 162500

Luis Julián Álvarez Armenta – 163247

Homework No. 3

Automotive Networks

02/05/22

Spring 2022

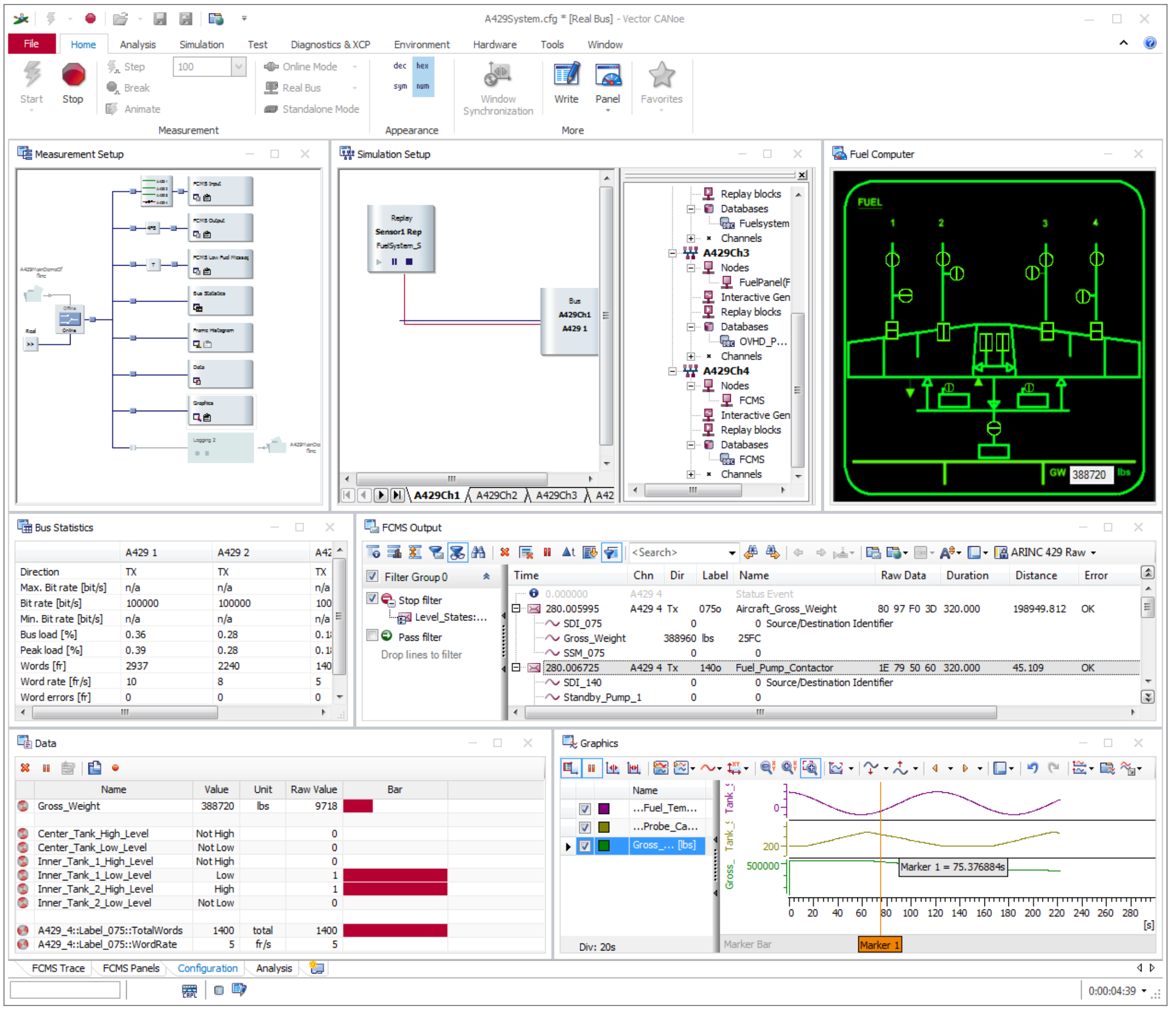
Abstract

In this work, the simulation of ECU nodes in the software CANoe is done. By creating configurations in CANoe together with panels, allowing the user to control some characteristics of a car, such as the doors or windows and visualize the status of a car, such as the ignition state.

On the other hand, an indicator system of a car is also done by using CAPL, where messages, environment variables and timers are considered in order to design nodes as part of the indicator system. Also, by creating databases and panels, the proper way of simulating this indictor system is achieved.

Introduction

The main tool used for this task is the CANoe software, where CANoe is a software mainly used to analyze and develop networks (and also individually) of ECUs (Electronic Control Unit) which are the devices in charge of controlling the functions of an automobile (by sending messages containing signals) (Vector, 2021). In other words, and as the name of the software says, CANoe allows modeling and simulating a CAN communication network. Figure 1 shows an example of the CANoe interface.



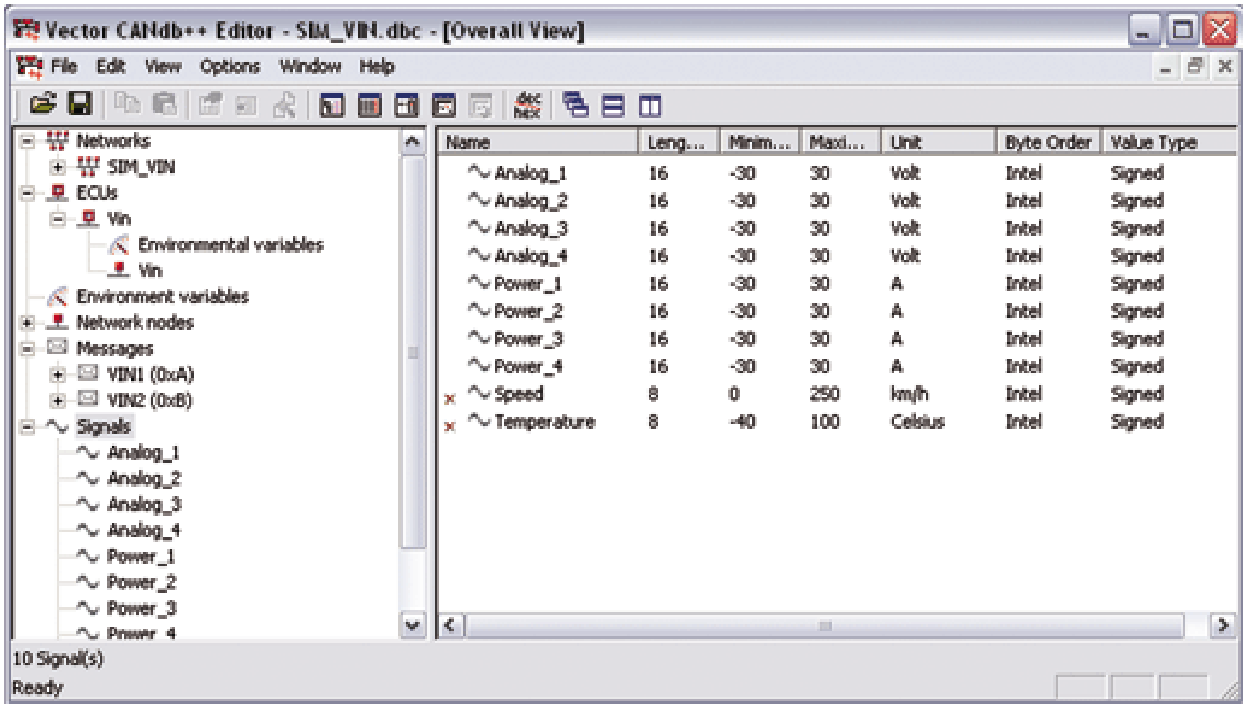
	A429 1	A429 2	A42
Direction	TX	TX	TX
Max. Bit rate [bit/s]	n/a	n/a	n/a
Bit rate [bit/s]	100000	100000	100
Min. Bit rate [bit/s]	n/a	n/a	n/a
Bus load [%]	0.36	0.28	0.1
Peak load [%]	0.39	0.28	0.1
Words [fr]	2937	2240	140
Word rate [fr/s]	10	8	5
Word errors [fr]	0	0	0

```
Time         Chn     Dir  Label  Name                                  Raw Data     Duration  Distance    Error
0.000000     A429 4               Status Event
280.005995   A429 4  Tx   075o   Aircraft_Gross_Weight                 80 97 F0 3D  320.000   198949.812  OK
  SDI_075              0          0 Source/Destination Identifier
  Gross_Weight    388960 lbs      25FC
  SSM_075              0          0
280.006725   A429 4  Tx   140o   Fuel_Pump_Contactor                   1E 79 50 60  320.000   45.109      OK
  SDI_140              0          0 Source/Destination Identifier
  Standby_Pump_1       0          0
```

Name	Value	Unit	Raw Value	Bar
Gross_Weight	388720	lbs	9718	
Center_Tank_High_Level	Not High		0	
Center_Tank_Low_Level	Not Low		0	
Inner_Tank_1_High_Level	Not High		0	
Inner_Tank_1_Low_Level	Low		1	
Inner_Tank_2_High_Level	High		1	
Inner_Tank_2_Low_Level	Not Low		0	
A429_4::Label_075::TotalWords	1400	total	1400	
A429_4::Label_075::WordRate	5	fr/s	5	

Figure 1. CANoe's interface. Retrieved from: https://www.vector.com/.

On the other hand, as mentioned on the website of the CANoe software developers, Vector (2021), the databases allow describing the ECUs connected to the bus and the messages and signals they transmit, as well as describing the properties of the CAN network itself. In this way, CANdb++ allows the creation, visualization and modification of these databases, in order to be able to simulate and analyze bus communication. Figure 2 shows an example of a CANdb++ database.



- Networks
  - SIM_VIN
- ECUs
  - Vin
    - Environmental variables
    - Vin
- Environment variables
- Network nodes
- Messages
  - VIN1 (0xA)
  - VIN2 (0xB)
- Signals
  - Analog_1
  - Analog_2
  - Analog_3
  - Analog_4
  - Power_1
  - Power_2
  - Power_3
  - Power_4

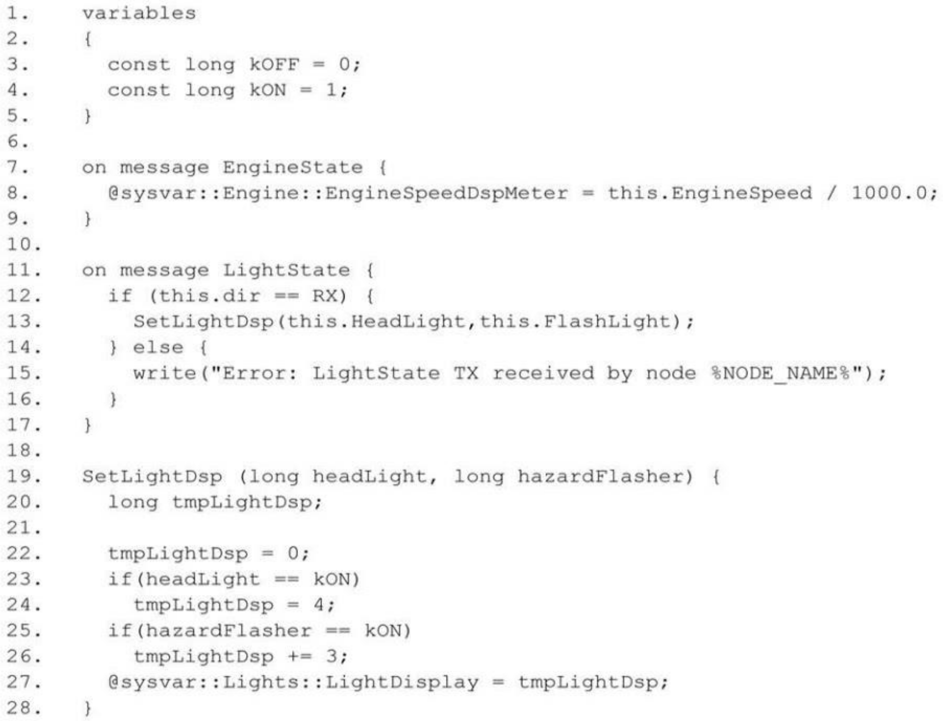
	Name	Leng...	Minim...	Maxi...	Unit	Byte Order	Value Type
	Analog_1	16	-30	30	Volt	Intel	Signed
	Analog_2	16	-30	30	Volt	Intel	Signed
	Analog_3	16	-30	30	Volt	Intel	Signed
	Analog_4	16	-30	30	Volt	Intel	Signed
	Power_1	16	-30	30	A	Intel	Signed
	Power_2	16	-30	30	A	Intel	Signed
	Power_3	16	-30	30	A	Intel	Signed
	Power_4	16	-30	30	A	Intel	Signed
×	Speed	8	0	250	km/h	Intel	Signed
×	Temperature	8	-40	100	Celsius	Intel	Signed

10 Signal(s)
Ready

Figure 2. Example of CANdb++ database. Retrieved from: https://electronics.stackexchange.com/.

Finally, as the last important concept for this work is CAPL used to create the emulation of a CAN network. As mentioned in the website "EMBEDDEDC.in", CAPL is a programming language similar to the C language, with the difference that CAPL is a procedural language, that is, the execution of each block of the program is event-driven. Thus, the code defined in the event procedure section is executed when a specific event occurs. In Figure 3, an example of an event procedure is shown.

In the case of this work, the buttons created in the panels that emulate the physical buttons used to control the locking/unlocking of doors or the opening of the windows of a car, are the events that trigger the code fragments.



```
variables
{
  const long kOFF = 0;
  const long kON = 1;
}

on message EngineState {
  @sysvar::Engine::EngineSpeedDspMeter = this.EngineSpeed / 1000.0;
}

on message LightState {
  if (this.dir == RX) {
    SetLightDsp(this.HeadLight,this.FlashLight);
  } else {
    write("Error: LightState TX received by node %NODE_NAME%");
  }
}

SetLightDsp (long headLight, long hazardFlasher) {
  long tmpLightDsp;

  tmpLightDsp = 0;
  if(headLight == kON)
    tmpLightDsp = 4;
  if(hazardFlasher == kON)
    tmpLightDsp += 3;
  @sysvar::Lights::LightDisplay = tmpLightDsp;
}
```

Figure 3. Example of code fragment programmed with CAPL. Retrieved from: https://www.vector.com/.

Methodology

In order to solve the problems, the following methodology was followed in each problem:

- **Part 1: Simulation of two ECU Nodes (Main Controls and Doors) in CANoe**

Create a complete CANoe configuration with two ECU network nodes models (Main Control and Doors) and associated control panels. The panel should give the user the ability to set the ignition to on/off, the doors to lock/unlock, and the window to up/down. We need to create five messages: IgnitionStatus, LockingControlRq, LockingSysState, WindowRq, and WindowState. Altogether, the following data is required:

CAN Messages

MainControl

1. Name = IgnitionStatus; DLC = 1.

2. Name = LockingControlRq; DLC = 1.

3. Name = WindowRq; DLC =1.

Doors

1. Name = LockingSysState; DLC = 1.

2. Name = WindowState; DLC = 2.

CAN Signals

1. Signal Name: Ignition_Status; Length=1; Minimum=0; Maximum=1; Unsigned. a. Message=IgnitionStatus.

2. Signal Name: LockRequest; Length=1; Minimum=0; Maximum=1; Unsigned. a. Message=LockingControlRq.

3. Signal Name: WindowPosition; Length=8; Minimum=0; Maximum=15; Unsigned. a. Message = WindowState.

4. Signal Name: LockState; Length=1; Minimum=0; Maximum=1; Unsigned. a. Message=LockingSysState.

5. Signal Name: WindowStatus; Length = 2; Minimum=0; Maximum=2; Unsigned. a. Message=WindowRq.

Environment variables

1. IgnitionStart: Type = Integer; Access = ReadWrite; Maximum value = 0x1

2. LockRq: Type = Integer; Access = ReadWrite; Maximum value = 0x1

3. Window: Type = Integer; Access = ReadWrite; Maximum value = 15

4. WindowDown: Type = Integer; Access = ReadWrite; Maximum value = 0x1

5. WindowUp: Type = Integer; Access = ReadWrite; Maximum value = 0x1

- **Part 2: Simulation of five ECU Nodes in CANoe**

The indicator system of a car shall be realized using a CAN-based distributed body electronic system. Because of the distance, each indicator lamp must be connected to another node (see Figure 1). The indicator lamps of the system shall indicate "turn left", "turn right" or "alarm" (all lamps blink). If the Demo application does not allow this number of ECUs, all these indicator lamps can be combined into one/two ECU. The blinking frequency shall be 1 Hz. The transmission rate shall be 80 kBaud.

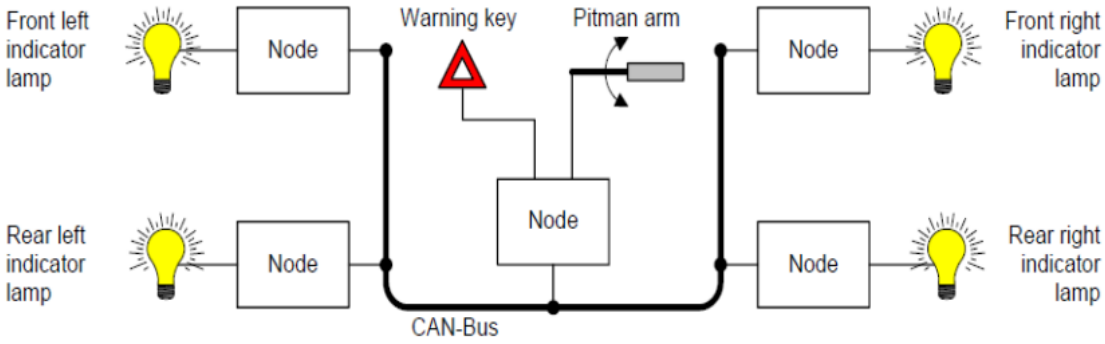

Front left indicator lamp
Node
Warning key
Pitman arm
Node
Front right indicator lamp
Node
Rear left indicator lamp
Node
Node
Rear right indicator lamp
CAN-Bus

Figure 4. Indicator system.

Task 1: Which CAN messages and environment variables are required to model this system?

For each CAN message, the following data are to be fixed:

- Name of the CAN message

- Identifier (maximum 11 Bit)

- Data-Length-Code

- Per signal: Name, number of bits, start bit

For environment variables, the following data are to be fixed:

- Name of the environment variable

- Data type of the environment variable

Task 2: Design a CAPL model for each node of the indicator system. Use the messages and environment variables determined in Task 1. Also use a timer to realize the blink frequency. You can use the idea of the following timer:

```
variables {
            timer delayTimer;
            message Msg1 sendMsg;
}
on start {
            setTimer(delayTimer, 2);
}
```

```
on timer delayTimer {
            output(sendMsg);
            setTimer(delayTimer, 2);
}
on envVar evSwitch {
            sendMsg.bsSwitch = getValue(this);
}
```

Figure 5. Timer example.

The timer works as follows. Using the event procedure “on start” the timer is initialized to the beginning of the simulation with the value 2 (seconds). As soon as the timer ran off, the event procedure “on timer delayTimer” is called, which outputs the CAN message “sendMsg” (resp. “Msg1”) with the current value of the signal “bsSwitch” and re-initializes the timer afterwards.

Experiment 1: Model and simulate the indicator system with CANoe. Proceed as follows:

1. Create a database based on the CAN messages and environment variables defined in task 1.

2. Create panels to visualize the lamps and to represent both the pitman arm and the warning key.

3. Realize the CAPL models designed in task 2.

4. Configure and perform a simulation.

Coupling the Model Process “Car Door”

As we did in Part 1, the produced model shall now also control the window lifter of the car door. The control itself can be integrated in the centrally represented node in Figure 4.

Task 3: Enlarge your CAPL model of the central node from Task 2, so that it can control the car door. Think first which further environment variables are required (e.g. to model the window lifter keys, see Part 1). The CAN messages to be used are already fixed by the interface of the car door (see Table 1). Take storing of and moving to certain window position as well as mirror positions into account in your CAPL model.

Experiment 2: Realize the supplements designed in Task 3. For this, enlarge the database of experiment 1 in accordance to your results of Task 3 (you will need some additional CAN messages and environment variables to control the door). Integrate the designed extensions of the central node into the corresponding model from experiment 1. Create panels to model the window lifter keys. Perform a remaining bus simulation with the car door connected. The car door can be controlled using the following CAN messages. As transmission rate, 80 kBaud is to be se

Results

Video showing the functioning of this homework is at: https://web.microsoftstream.com/video/21ad5f90-f5a8-451f-a85d-36ade912b909

- **Part 1**

Figure 6, shows the resultant ECU network nodes.

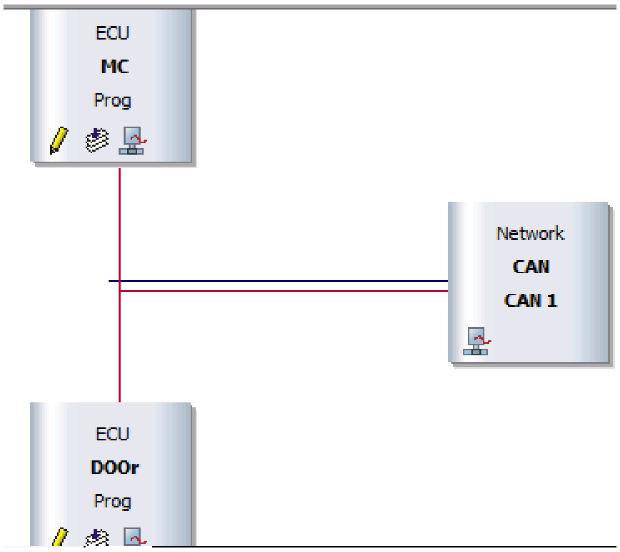


Figure 6. ECU network nodes.

Figure 7, shows the designed panel where a button to set the ignition on/off and an indicator of this status is added to the panel. Also, another button to lock/unlock the doors with its light indicator are added and finally, two buttons to move the windows up and down (labeled as such, respectively) and a needle indicator (to show the opening percentage of the windows) are added to the panel as well.

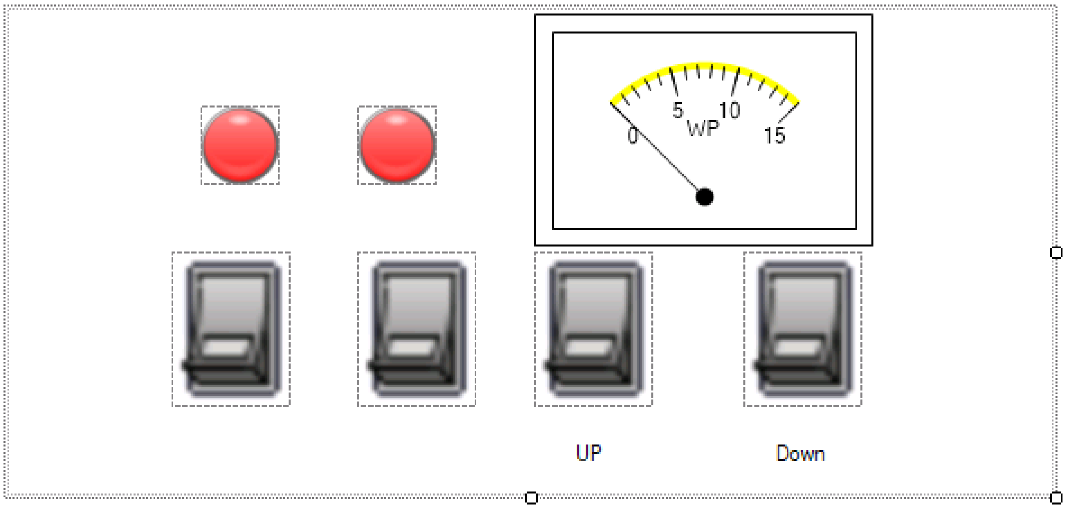
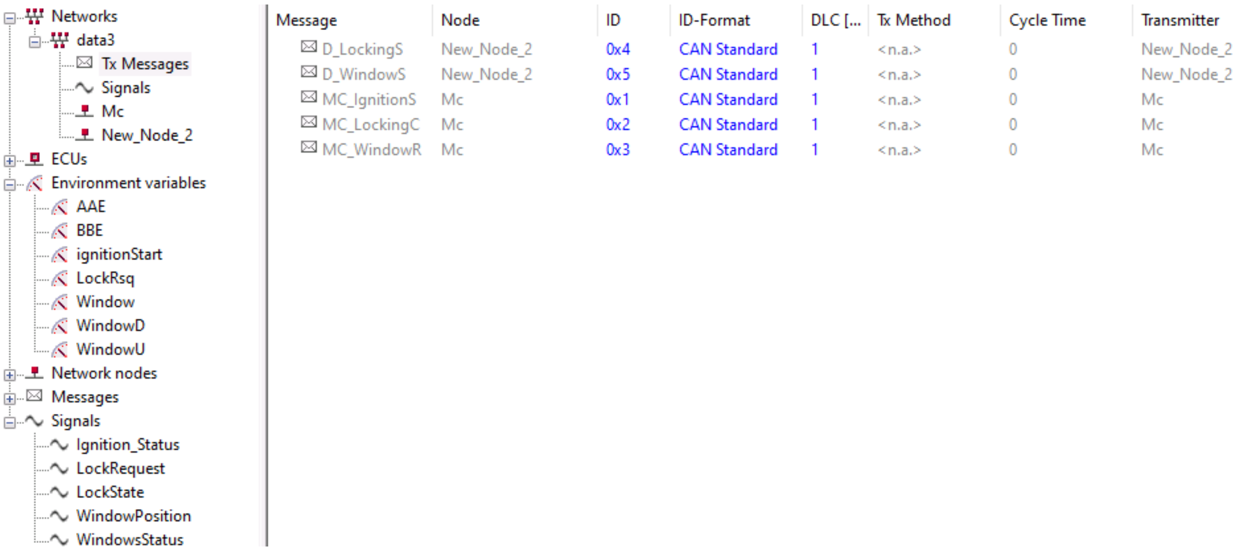


Figure 7. ECU network nodes.

Figure 8, shows the database created with the messages, signals and environmental variables requested.

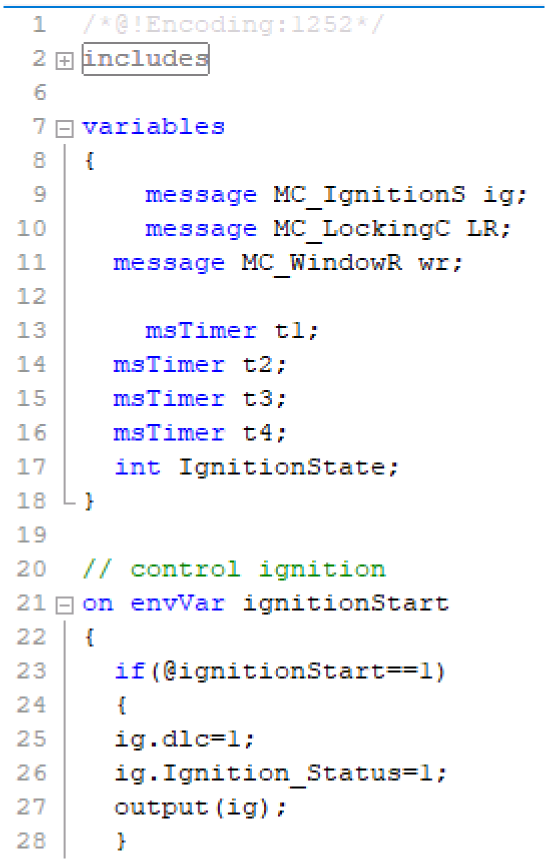


- Networks
  - data3
    - Tx Messages
    - Signals
    - Mc
    - New_Node_2
- ECUs
- Environment variables
  - AAE
  - BBE
  - ignitionStart
  - LockRsq
  - Window
  - WindowD
  - WindowU
- Network nodes
- Messages
- Signals
  - Ignition_Status
  - LockRequest
  - LockState
  - WindowPosition
  - WindowsStatus

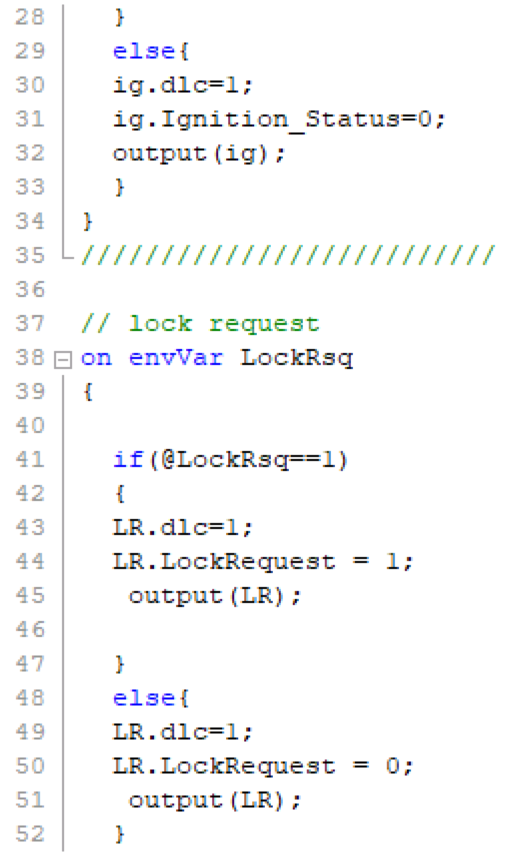
Message	Node	ID	ID-Format	DLC [...	Tx Method	Cycle Time	Transmitter
D_LockingS	New_Node_2	0x4	CAN Standard	1	<n.a.>	0	New_Node_2
D_WindowS	New_Node_2	0x5	CAN Standard	1	<n.a.>	0	New_Node_2
MC_IgnitionS	Mc	0x1	CAN Standard	1	<n.a.>	0	Mc
MC_LockingC	Mc	0x2	CAN Standard	1	<n.a.>	0	Mc
MC_WindowR	Mc	0x3	CAN Standard	1	<n.a.>	0	Mc

Figure 8. CANdb for this ECU network.

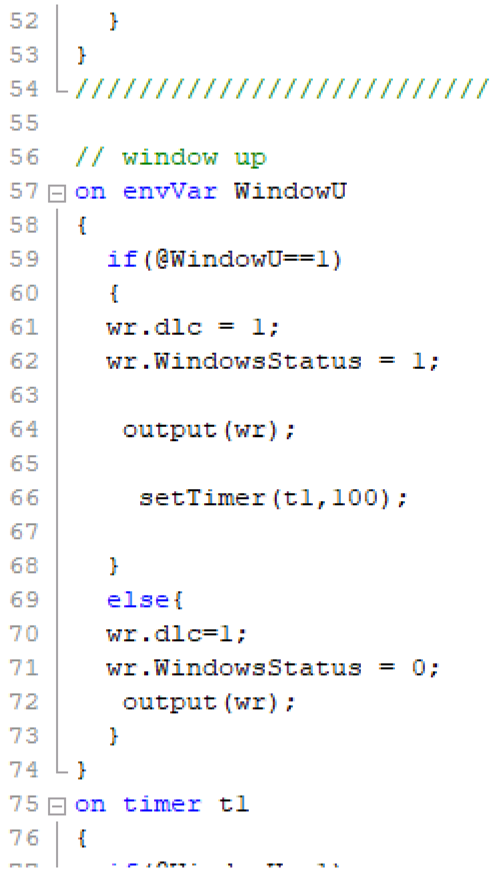
Figure 9, shows the CAPL code implemented for node “MC” which stands for “Main Controls”. Here, the ignitions status, the locking/unlocking of doors and windows up and down movement events, are controlled.



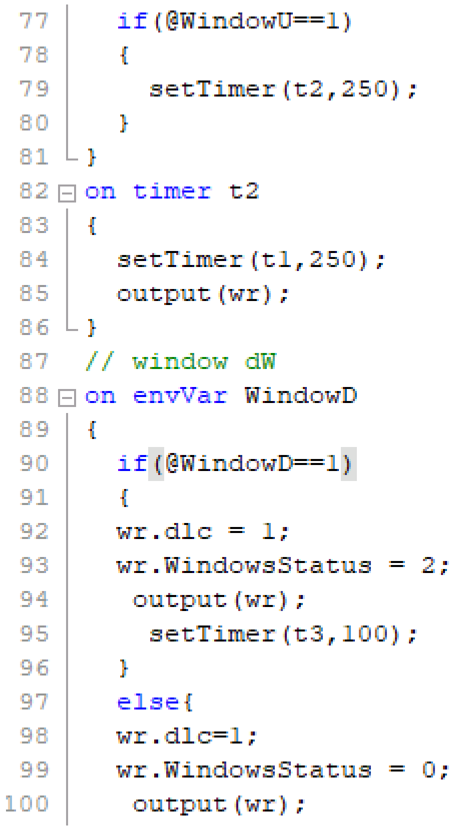
```
/*@!Encoding:1252*/
includes

variables
{
    message MC_IgnitionS ig;
    message MC_LockingC LR;
  message MC_WindowR wr;

    msTimer t1;
  msTimer t2;
  msTimer t3;
  msTimer t4;
  int IgnitionState;
}

// control ignition
on envVar ignitionStart
{
  if(@ignitionStart==1)
  {
  ig.dlc=1;
  ig.Ignition_Status=1;
  output(ig);
  }
```



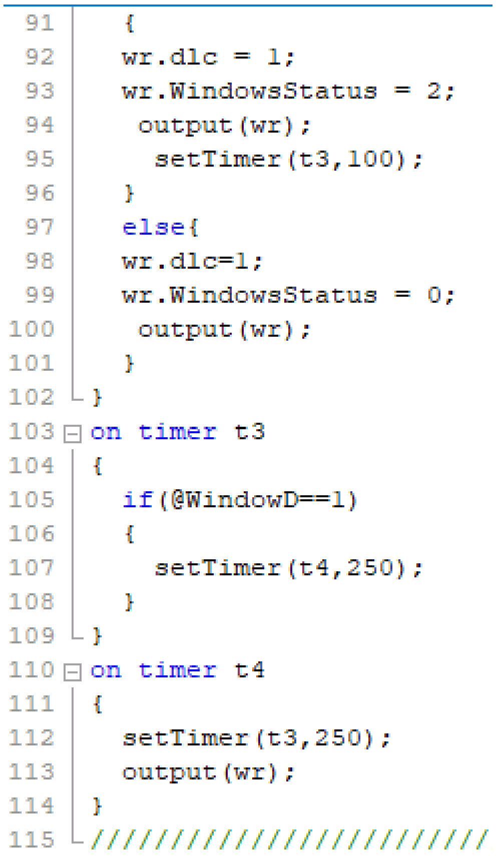
```
  }
  else{
  ig.dlc=1;
  ig.Ignition_Status=0;
  output(ig);
  }
}
//////////////////////////

// lock request
on envVar LockRsq
{

  if(@LockRsq==1)
  {
  LR.dlc=1;
  LR.LockRequest = 1;
   output(LR);

  }
  else{
  LR.dlc=1;
  LR.LockRequest = 0;
   output(LR);
  }
```



```
   }
}
//////////////////////////

// window up
on envVar WindowU
{
  if(@WindowU==1)
  {
  wr.dlc = 1;
  wr.WindowsStatus = 1;

   output(wr);

    setTimer(t1,100);

  }
  else{
  wr.dlc=1;
  wr.WindowsStatus = 0;
   output(wr);
  }
}
on timer t1
{
```



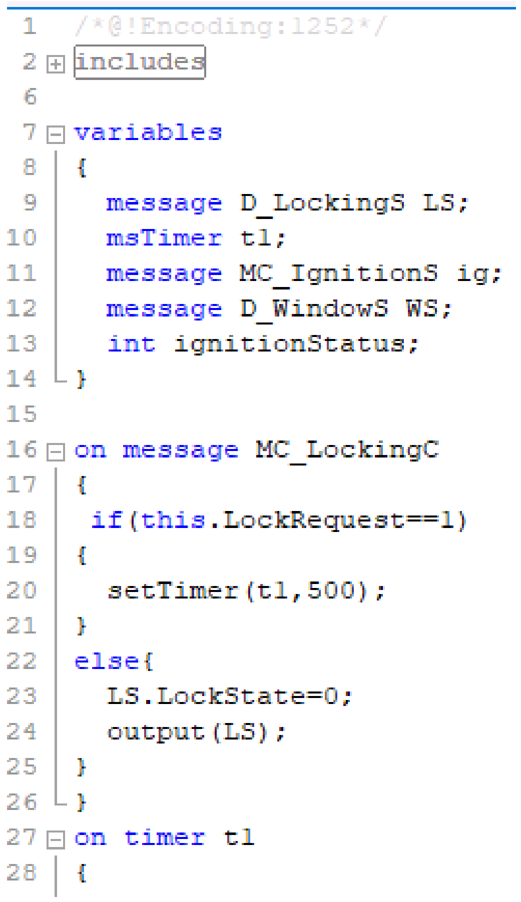
```
  if(@WindowU==1)
  {
    setTimer(t2,250);
  }
}
on timer t2
{
  setTimer(t1,250);
  output(wr);
}
// window dW
on envVar WindowD
{
  if(@WindowD==1)
  {
  wr.dlc = 1;
  wr.WindowsStatus = 2;
   output(wr);
    setTimer(t3,100);
  }
  else{
  wr.dlc=1;
  wr.WindowsStatus = 0;
   output(wr);
```



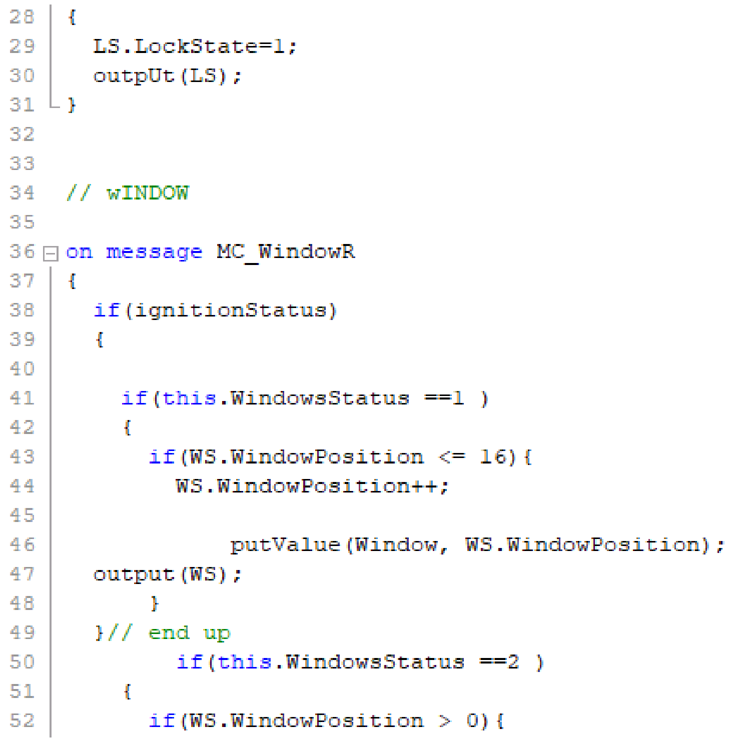
```
  {
  wr.dlc = 1;
  wr.WindowsStatus = 2;
   output(wr);
    setTimer(t3,100);
  }
  else{
  wr.dlc=1;
  wr.WindowsStatus = 0;
   output(wr);
  }
}
on timer t3
{
  if(@WindowD==1)
  {
    setTimer(t4,250);
  }
}
on timer t4
{
  setTimer(t3,250);
  output(wr);
}
/////////////////////////
```

Figure 9. CAPL code for node “MC”.

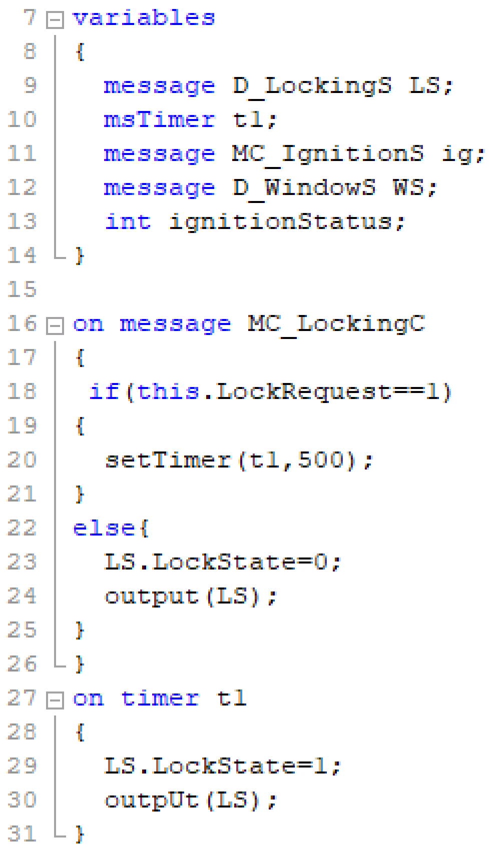
Figure 9.1, shows the CAPL code implemented for node “DOOr”. Here, the code to show in the needle indicator the position of the windows, the red light indicators to show that the ignition is set to on and the doors are locked is implemented.



```
/*@!Encoding:1252*/
includes

variables
{
  message D_LockingS LS;
  msTimer t1;
  message MC_IgnitionS ig;
  message D_WindowS WS;
  int ignitionStatus;
}

on message MC_LockingC
{
 if(this.LockRequest==1)
{
  setTimer(t1,500);
}
else{
  LS.LockState=0;
  output(LS);
}
}
on timer t1
{
```



```
{
  LS.LockState=1;
  outpUt(LS);
}


// wINDOW

on message MC_WindowR
{
  if(ignitionStatus)
  {

    if(this.WindowsStatus ==1 )
    {
      if(WS.WindowPosition <= 16){
        WS.WindowPosition++;

            putValue(Window, WS.WindowPosition);
  output(WS);
      }
  }// end up
        if(this.WindowsStatus ==2 )
    {
      if(WS.WindowPosition > 0){
```



```
variables
{
  message D_LockingS LS;
  msTimer t1;
  message MC_IgnitionS ig;
  message D_WindowS WS;
  int ignitionStatus;
}

on message MC_LockingC
{
 if(this.LockRequest==1)
{
  setTimer(t1,500);
}
else{
  LS.LockState=0;
  output(LS);
}
}
on timer t1
{
  LS.LockState=1;
  outpUt(LS);
}
```

Figure 9.1. CAPL code for node “DOOr”.

Next figures show the functioning of the panel designed to control the characteristics of the car.

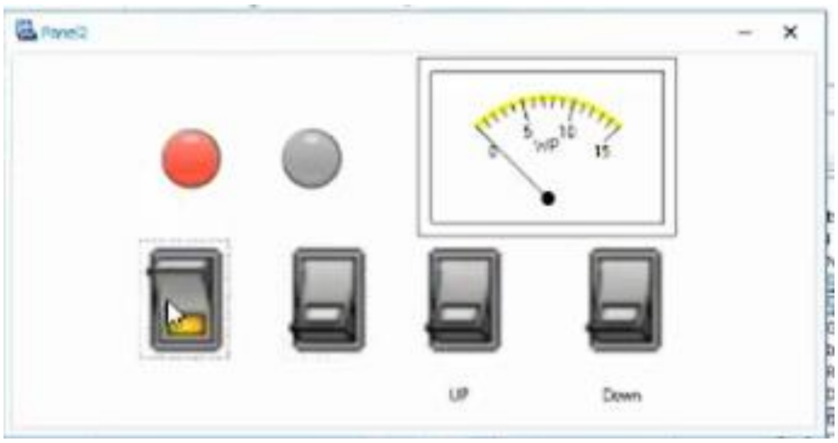


Figure 10. Ignition set to on with red light indicator showing so and doors unlocked with indicator turned off.

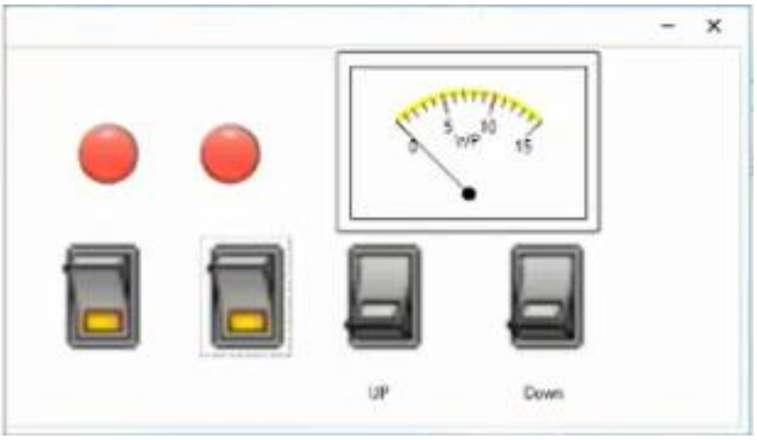


Figure 11. Ignition set to on with red light indicator showing so and door locked with red light indicator turned on.

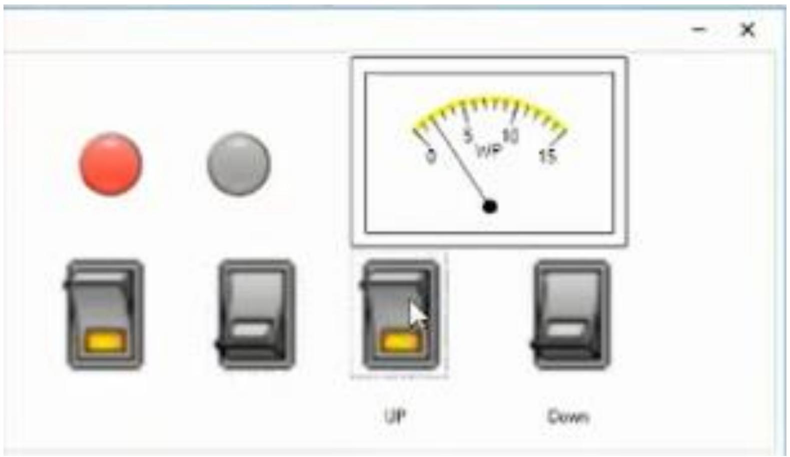


Figure 12. Ignition set to on with red light indicator showing so and window moving up with needle window position indicator showing so.

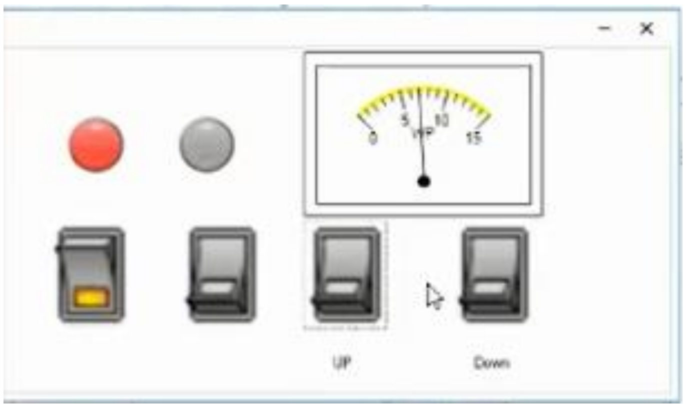


Figure 13. Ignition set to on with red light indicator showing so and window moving up and down with needle window position indicating a halfway position.

Figure 14, shows how if the ignition status is set to “off”, no button will work due to this (like in this case that the window is not moving up despite the button being pressed).

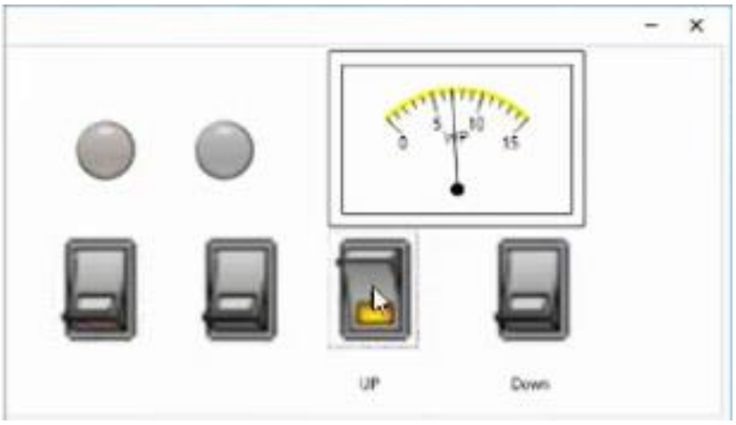


Figure 14. Ignition set to off with red light indicator turned off and windows not moving up.

- **Part 2**

Figure 15, shows the five ECU network implemented for the indicator system of the car which control the indicator lamps (nodes names “nodo1”, “node2v2, “nodo3” and “nodo4”), the door controls and the main controls.

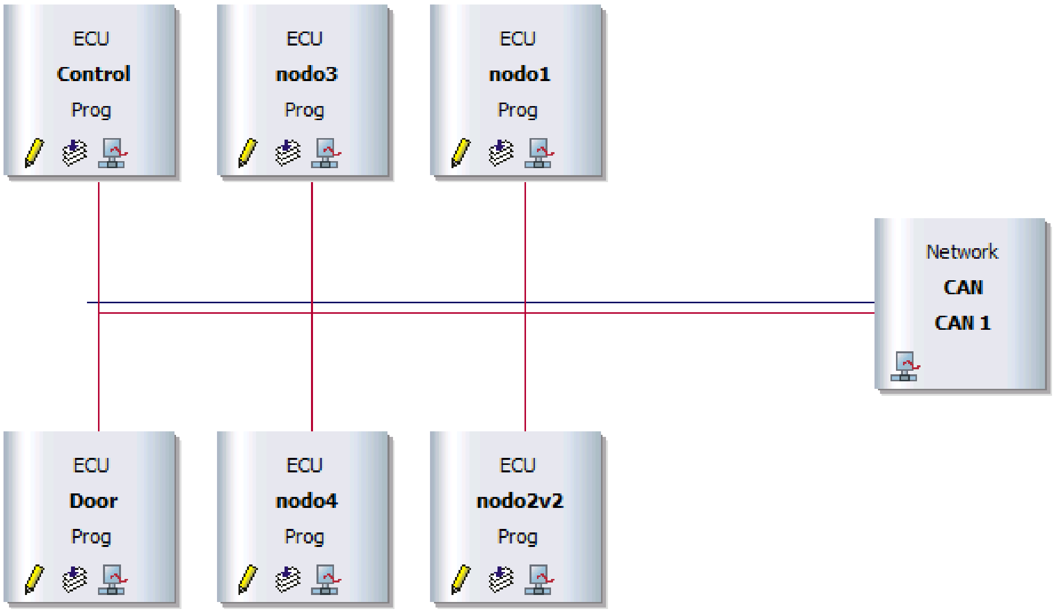


Figure 15. ECU network for the indicator system.

Figure 16, shows the panel designed with all buttons implemented to control the indicator lamps (which represent the turn left or right action), the alarm indicator lamp, the window movement controllers (up & down), a button to save the current position of the windows and another one to go to that last window position saved. A button to block the movement of the windows is also added, so that if this button is turned on, even if the up and down buttons of the windows are pushed, the windows will not move.

There is also a similar interface of buttons for the positions of the mirrors where there are buttons to extend and retract the mirrors with a bar indicator of the position and also, a button to save the current mirror position and other buttons which take you to the last 3 positions saved. 4 other buttons were added in order to move the mirrors either, up, down, left or right. All these buttons just mentioned have their own indicators to let the user know visually which buttons are activated and in the case of the position of the windows and mirrors, there are needle indicators for the windows and bar indicators for the mirrors to know where these elements are located in terms of their position.

All buttons are labeled with their respective names so that they are easy to identify inside the panel.

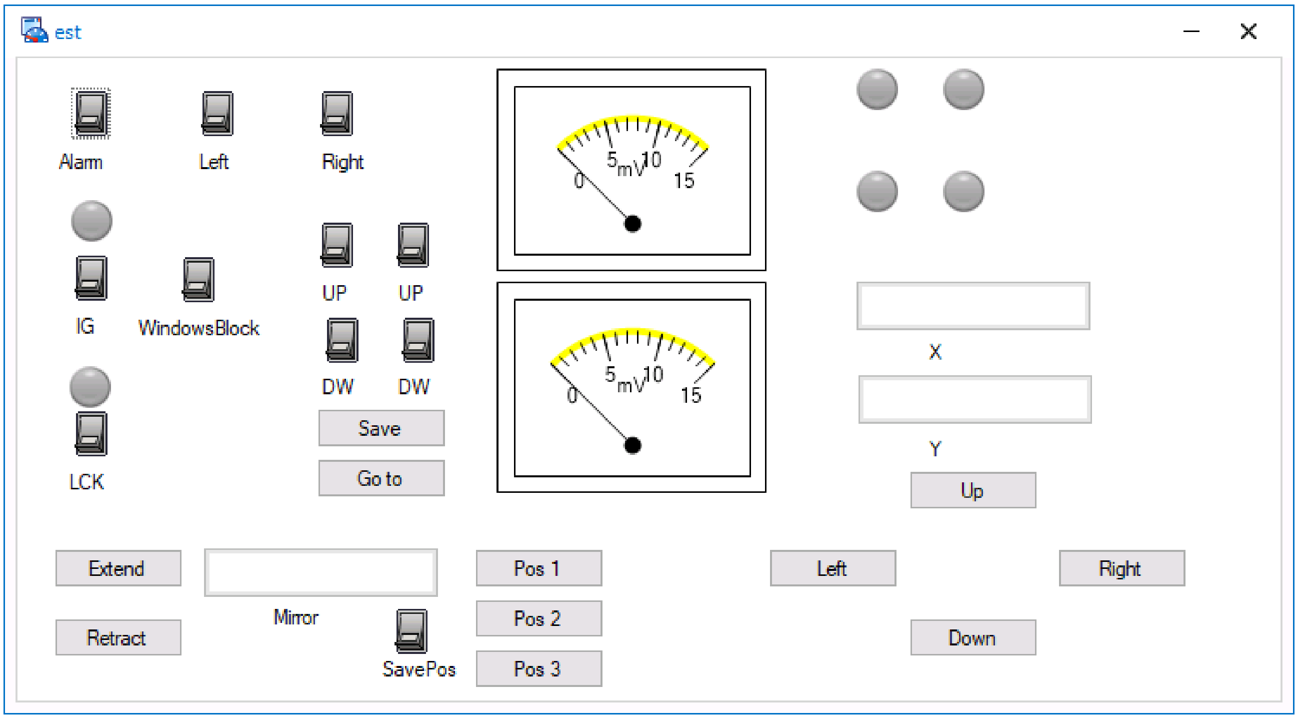
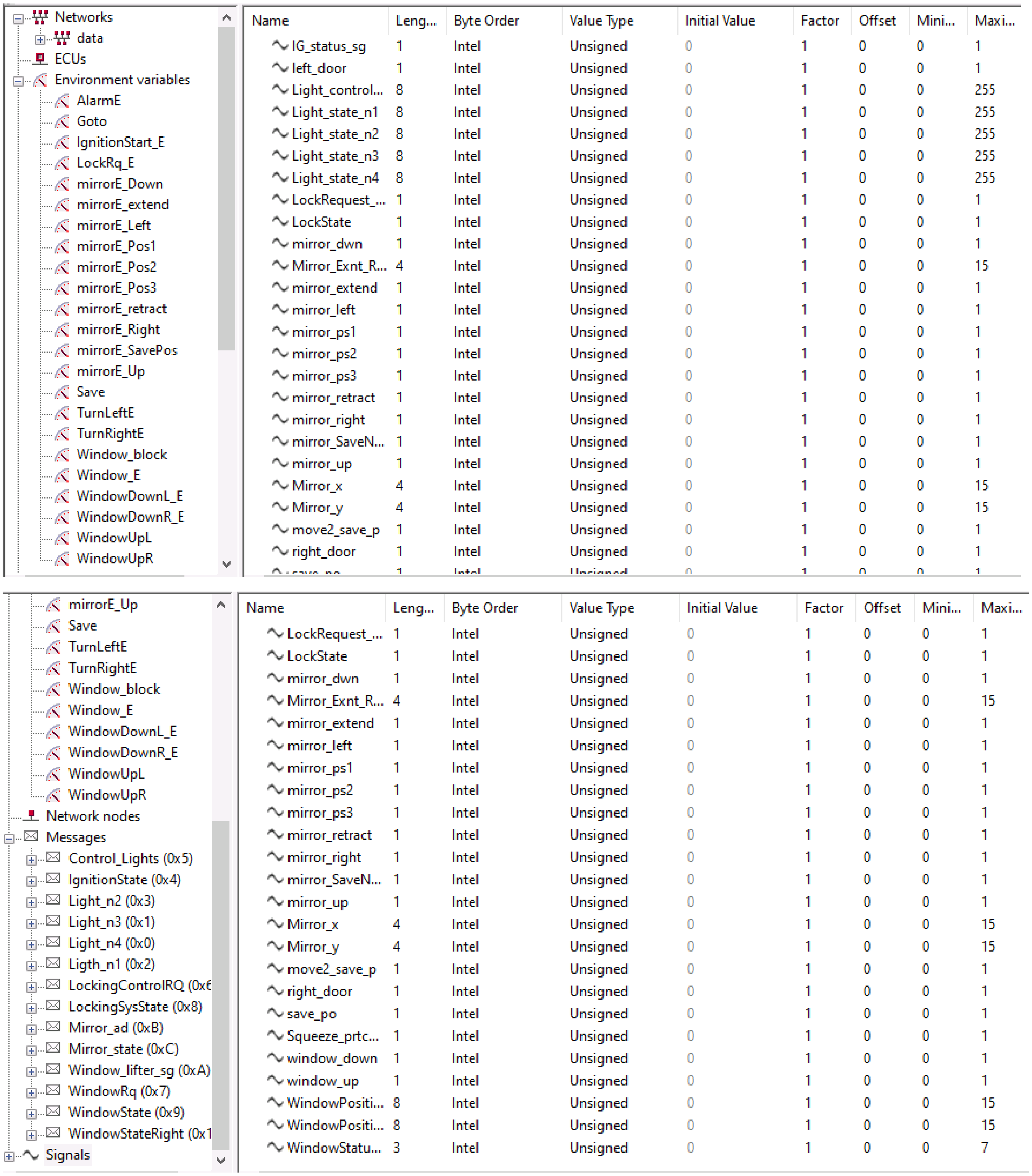


Figure 16. Panel designed for the indicator system.

Figure 17, shows the CANdb database created with all messages, signals and environmental variables required in order to make the proper functioning of the indicator system.



- Networks
  - data
- ECUs
- Environment variables
  - AlarmE
  - Goto
  - IgnitionStart_E
  - LockRq_E
  - mirrorE_Down
  - mirrorE_extend
  - mirrorE_Left
  - mirrorE_Pos1
  - mirrorE_Pos2
  - mirrorE_Pos3
  - mirrorE_retract
  - mirrorE_Right
  - mirrorE_SavePos
  - mirrorE_Up
  - Save
  - TurnLeftE
  - TurnRightE
  - Window_block
  - Window_E
  - WindowDownL_E
  - WindowDownR_E
  - WindowUpL
  - WindowUpR

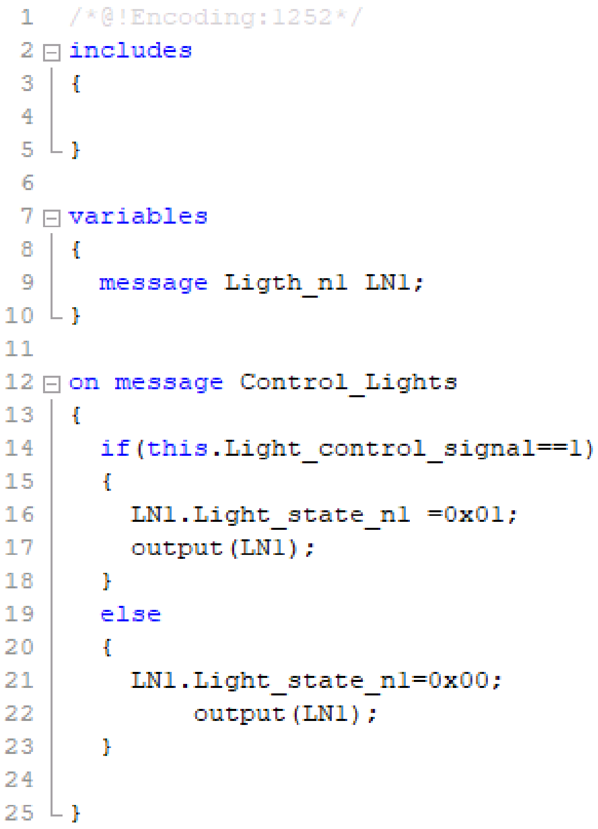
Name	Leng...	Byte Order	Value Type	Initial Value	Factor	Offset	Mini...	Maxi...
IG_status_sg	1	Intel	Unsigned	0	1	0	0	1
left_door	1	Intel	Unsigned	0	1	0	0	1
Light_control...	8	Intel	Unsigned	0	1	0	0	255
Light_state_n1	8	Intel	Unsigned	0	1	0	0	255
Light_state_n2	8	Intel	Unsigned	0	1	0	0	255
Light_state_n3	8	Intel	Unsigned	0	1	0	0	255
Light_state_n4	8	Intel	Unsigned	0	1	0	0	255
LockRequest_...	1	Intel	Unsigned	0	1	0	0	1
LockState	1	Intel	Unsigned	0	1	0	0	1
mirror_dwn	1	Intel	Unsigned	0	1	0	0	1
Mirror_Exnt_R...	4	Intel	Unsigned	0	1	0	0	15
mirror_extend	1	Intel	Unsigned	0	1	0	0	1
mirror_left	1	Intel	Unsigned	0	1	0	0	1
mirror_ps1	1	Intel	Unsigned	0	1	0	0	1
mirror_ps2	1	Intel	Unsigned	0	1	0	0	1
mirror_ps3	1	Intel	Unsigned	0	1	0	0	1
mirror_retract	1	Intel	Unsigned	0	1	0	0	1
mirror_right	1	Intel	Unsigned	0	1	0	0	1
mirror_SaveN...	1	Intel	Unsigned	0	1	0	0	1
mirror_up	1	Intel	Unsigned	0	1	0	0	1
Mirror_x	4	Intel	Unsigned	0	1	0	0	15
Mirror_y	4	Intel	Unsigned	0	1	0	0	15
move2_save_p	1	Intel	Unsigned	0	1	0	0	1
right_door	1	Intel	Unsigned	0	1	0	0	1
save_po	1	Intel	Unsigned	0	1	0	0	1

- mirrorE_Up
- Save
- TurnLeftE
- TurnRightE
- Window_block
- Window_E
- WindowDownL_E
- WindowDownR_E
- WindowUpL
- WindowUpR
- Network nodes
- Messages
  - Control_Lights (0x5)
  - IgnitionState (0x4)
  - Light_n2 (0x3)
  - Light_n3 (0x1)
  - Light_n4 (0x0)
  - Ligth_n1 (0x2)
  - LockingControlRQ (0x6
  - LockingSysState (0x8)
  - Mirror_ad (0xB)
  - Mirror_state (0xC)
  - Window_lifter_sg (0xA)
  - WindowRq (0x7)
  - WindowState (0x9)
  - WindowStateRight (0x1
- Signals

Name	Leng...	Byte Order	Value Type	Initial Value	Factor	Offset	Mini...	Maxi...
LockRequest_...	1	Intel	Unsigned	0	1	0	0	1
LockState	1	Intel	Unsigned	0	1	0	0	1
mirror_dwn	1	Intel	Unsigned	0	1	0	0	1
Mirror_Exnt_R...	4	Intel	Unsigned	0	1	0	0	15
mirror_extend	1	Intel	Unsigned	0	1	0	0	1
mirror_left	1	Intel	Unsigned	0	1	0	0	1
mirror_ps1	1	Intel	Unsigned	0	1	0	0	1
mirror_ps2	1	Intel	Unsigned	0	1	0	0	1
mirror_ps3	1	Intel	Unsigned	0	1	0	0	1
mirror_retract	1	Intel	Unsigned	0	1	0	0	1
mirror_right	1	Intel	Unsigned	0	1	0	0	1
mirror_SaveN...	1	Intel	Unsigned	0	1	0	0	1
mirror_up	1	Intel	Unsigned	0	1	0	0	1
Mirror_x	4	Intel	Unsigned	0	1	0	0	15
Mirror_y	4	Intel	Unsigned	0	1	0	0	15
move2_save_p	1	Intel	Unsigned	0	1	0	0	1
right_door	1	Intel	Unsigned	0	1	0	0	1
save_po	1	Intel	Unsigned	0	1	0	0	1
Squeeze_prtc...	1	Intel	Unsigned	0	1	0	0	1
window_down	1	Intel	Unsigned	0	1	0	0	1
window_up	1	Intel	Unsigned	0	1	0	0	1
WindowPositi...	8	Intel	Unsigned	0	1	0	0	15
WindowPositi...	8	Intel	Unsigned	0	1	0	0	15
WindowStatu...	3	Intel	Unsigned	0	1	0	0	7

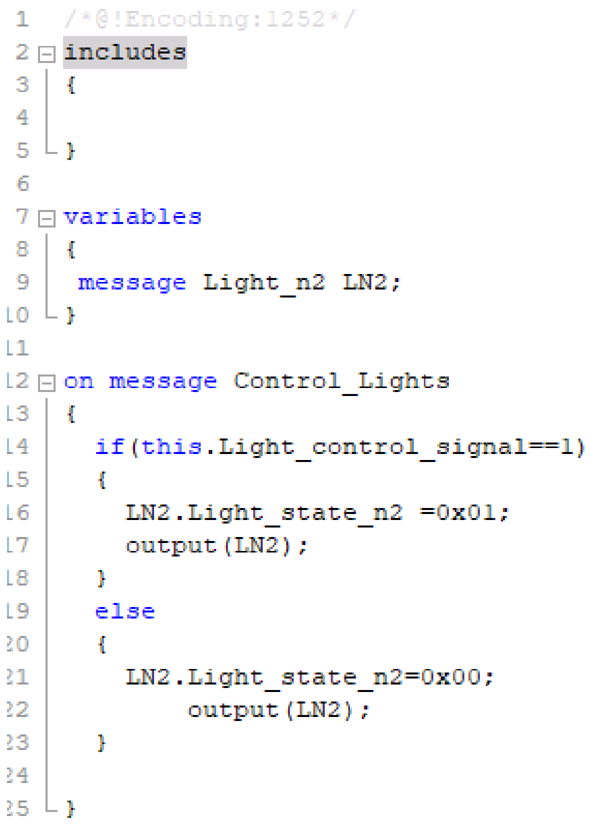
Figure 17. CANdb database for the indicator system.

Figure 18, shows the CAPL code for the “nodo1” node. *Figure 18.1*, shows the CAPL code for the “nodo2v2” node. *Figure 18.2*, shows the CAPL code for the “nodo3” node. *Figure 18.3*, shows the CAPL code for the “nodo4” node which are all the nodes use to control the indicator lamps.



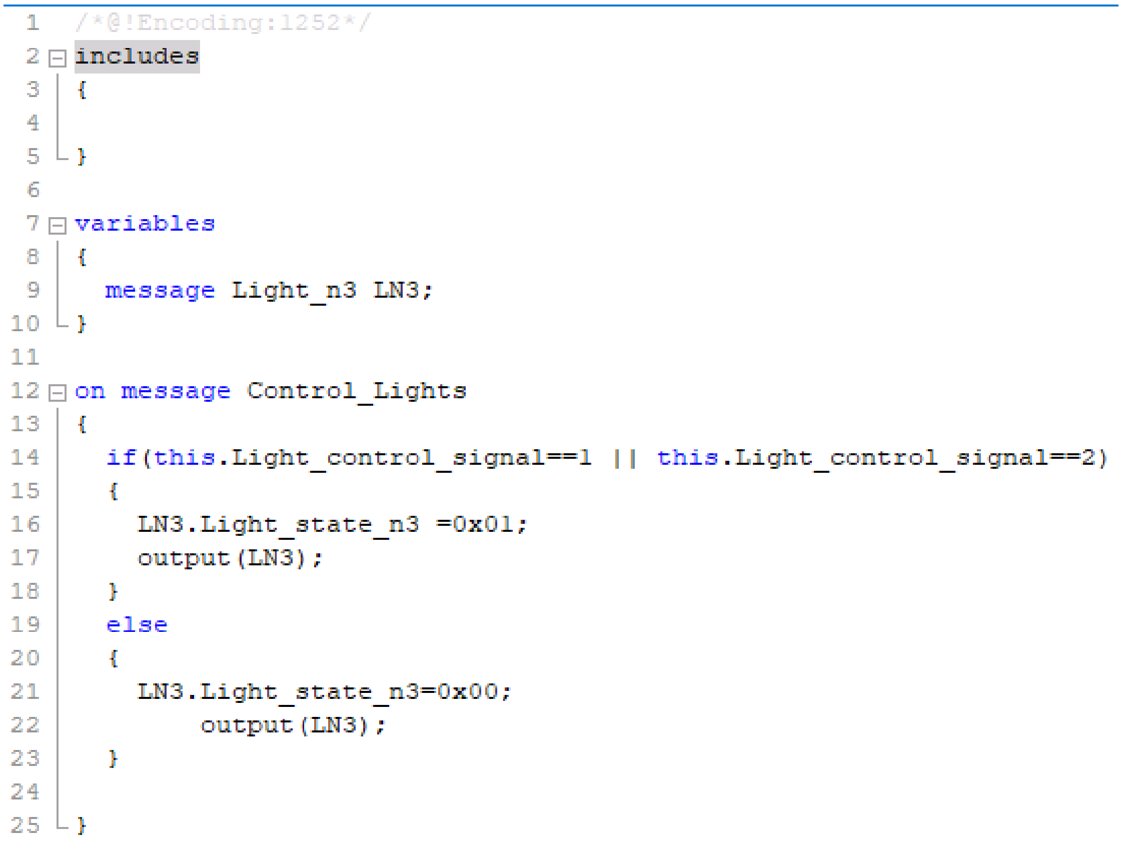
```
/*@!Encoding:1252*/
includes
{

}

variables
{
  message Ligth_n1 LN1;
}

on message Control_Lights
{
  if(this.Light_control_signal==1)
  {
    LN1.Light_state_n1 =0x01;
    output(LN1);
  }
  else
  {
    LN1.Light_state_n1=0x00;
        output(LN1);
  }

}
```

Figure 18. CAPL code for “nodo1”.



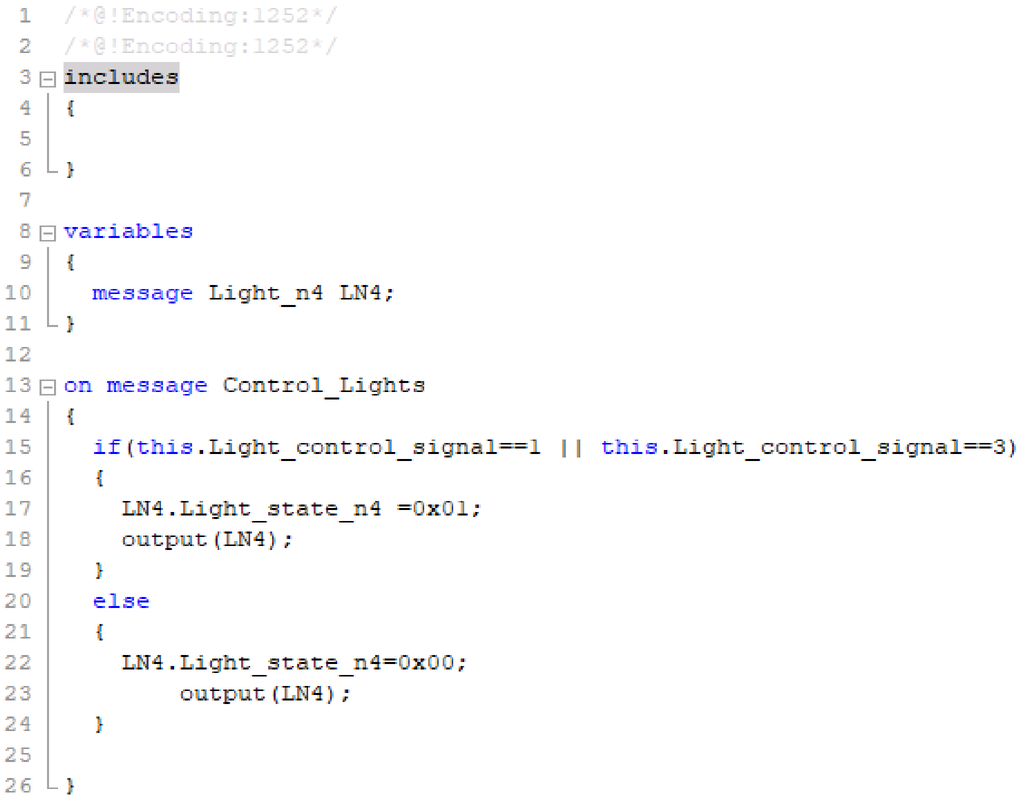
```
/*@!Encoding:1252*/
includes
{


}

variables
{
 message Light_n2 LN2;
}

on message Control_Lights
{
  if(this.Light_control_signal==1)
  {
    LN2.Light_state_n2 =0x01;
    output(LN2);
  }
  else
  {
    LN2.Light_state_n2=0x00;
        output(LN2);
  }

}
```

Figure 18.1. CAPL code for “nodo2v2”.



```
/*@!Encoding:1252*/
includes
{

}

variables
{
  message Light_n3 LN3;
}

on message Control_Lights
{
  if(this.Light_control_signal==1 || this.Light_control_signal==2)
  {
    LN3.Light_state_n3 =0x01;
    output(LN3);
  }
  else
  {
    LN3.Light_state_n3=0x00;
        output(LN3);
  }

}
```

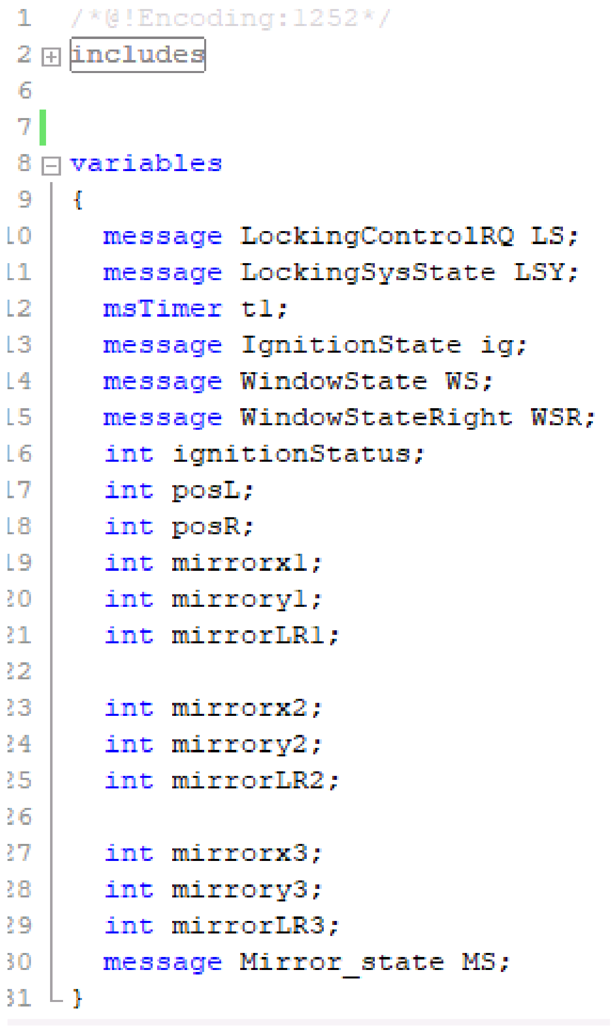
Figure 18.2. CAPL code for “nodo3”.



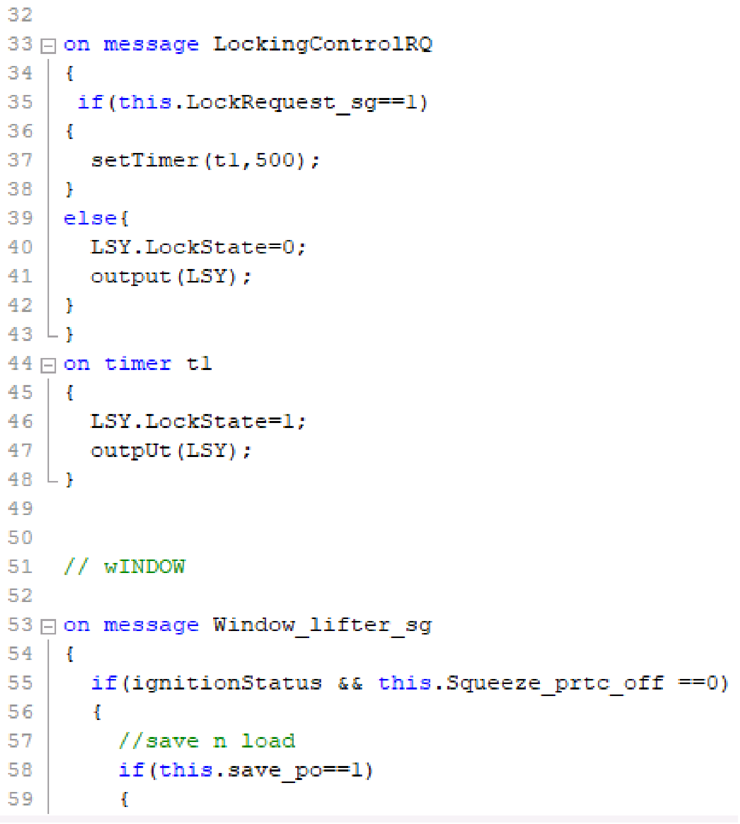
```
/*@!Encoding:1252*/
/*@!Encoding:1252*/
includes
{

}

variables
{
  message Light_n4 LN4;
}

on message Control_Lights
{
  if(this.Light_control_signal==1 || this.Light_control_signal==3)
  {
    LN4.Light_state_n4 =0x01;
    output(LN4);
  }
  else
  {
    LN4.Light_state_n4=0x00;
        output(LN4);
  }

}
```

Figure 18.3. CAPL code for “nodo4”.

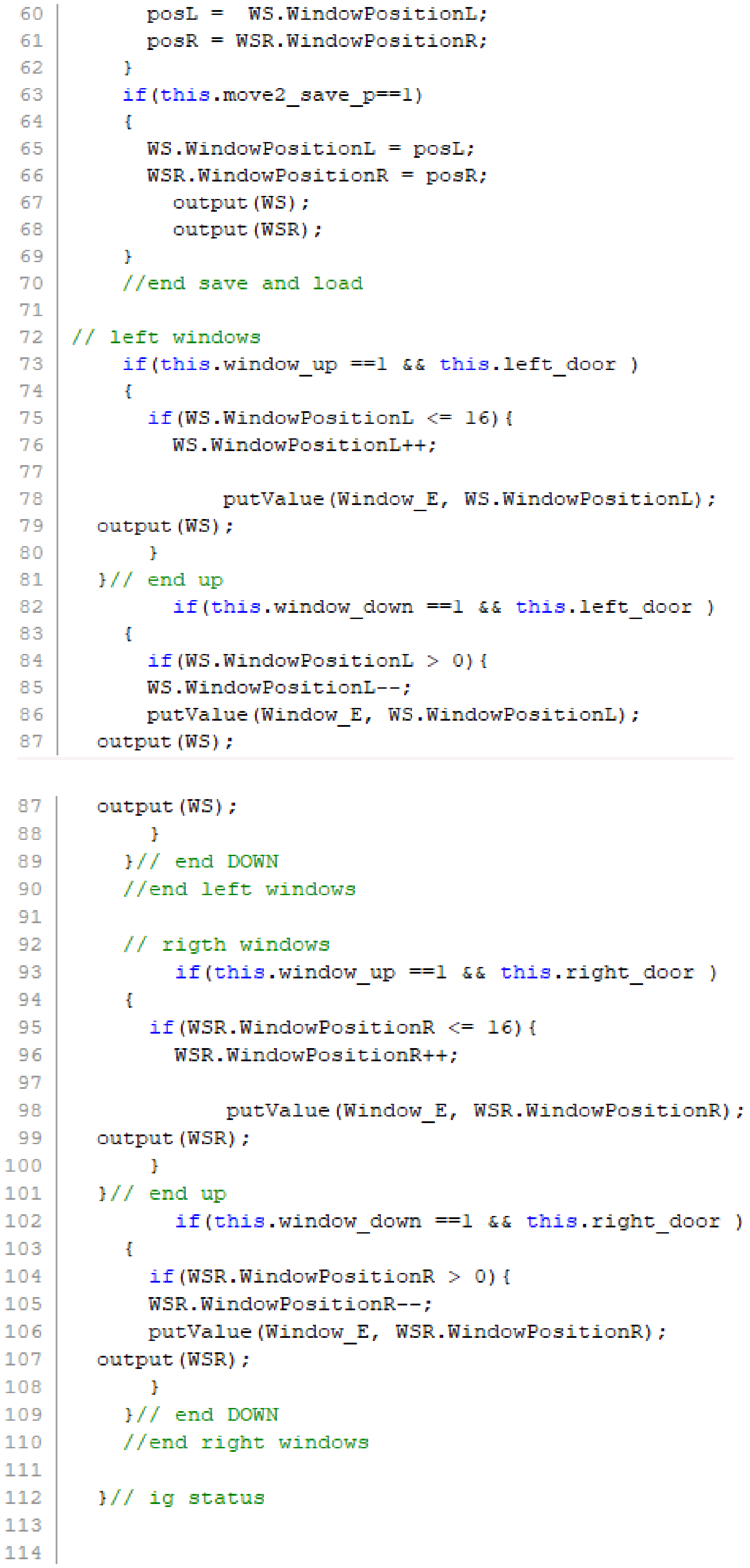
Figure 19, shows fragments of the CAPL code (due to the original code being more than 300 lines long) for node “Door” implemented to control the car functions such as the mirror and window positions with its saving and go to options.



```
/*@!Encoding:1252*/
includes


variables
{
  message LockingControlRQ LS;
  message LockingSysState LSY;
  msTimer t1;
  message IgnitionState ig;
  message WindowState WS;
  message WindowStateRight WSR;
  int ignitionStatus;
  int posL;
  int posR;
  int mirrorx1;
  int mirrory1;
  int mirrorLR1;

  int mirrorx2;
  int mirrory2;
  int mirrorLR2;

  int mirrorx3;
  int mirrory3;
  int mirrorLR3;
  message Mirror_state MS;
}
```



```

on message LockingControlRQ
{
 if(this.LockRequest_sg==1)
{
  setTimer(t1,500);
}
else{
  LSY.LockState=0;
  output(LSY);
}
}
on timer t1
{
  LSY.LockState=1;
  outpUt(LSY);
}


// wINDOW

on message Window_lifter_sg
{
  if(ignitionStatus && this.Squeeze_prtc_off ==0)
  {
    //save n load
    if(this.save_po==1)
    {
```

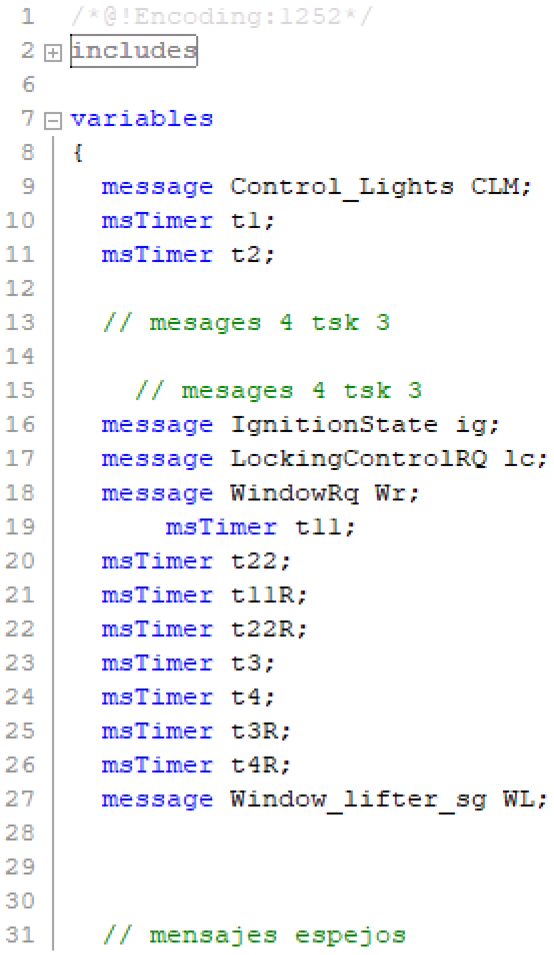


```
      posL =  WS.WindowPositionL;
      posR = WSR.WindowPositionR;
    }
    if(this.move2_save_p==1)
    {
      WS.WindowPositionL = posL;
      WSR.WindowPositionR = posR;
        output(WS);
        output(WSR);
    }
    //end save and load

// left windows
    if(this.window_up ==1 && this.left_door )
    {
      if(WS.WindowPositionL <= 16){
        WS.WindowPositionL++;

            putValue(Window_E, WS.WindowPositionL);
  output(WS);
      }
  }// end up
        if(this.window_down ==1 && this.left_door )
    {
      if(WS.WindowPositionL > 0){
      WS.WindowPositionL--;
      putValue(Window_E, WS.WindowPositionL);
  output(WS);
```

```
  output(WS);
      }
    }// end DOWN
    //end left windows

    // rigth windows
        if(this.window_up ==1 && this.right_door )
    {
      if(WSR.WindowPositionR <= 16){
        WSR.WindowPositionR++;

            putValue(Window_E, WSR.WindowPositionR);
  output(WSR);
      }
  }// end up
        if(this.window_down ==1 && this.right_door )
    {
      if(WSR.WindowPositionR > 0){
      WSR.WindowPositionR--;
      putValue(Window_E, WSR.WindowPositionR);
  output(WSR);
      }
    }// end DOWN
    //end right windows

  }// ig status


```

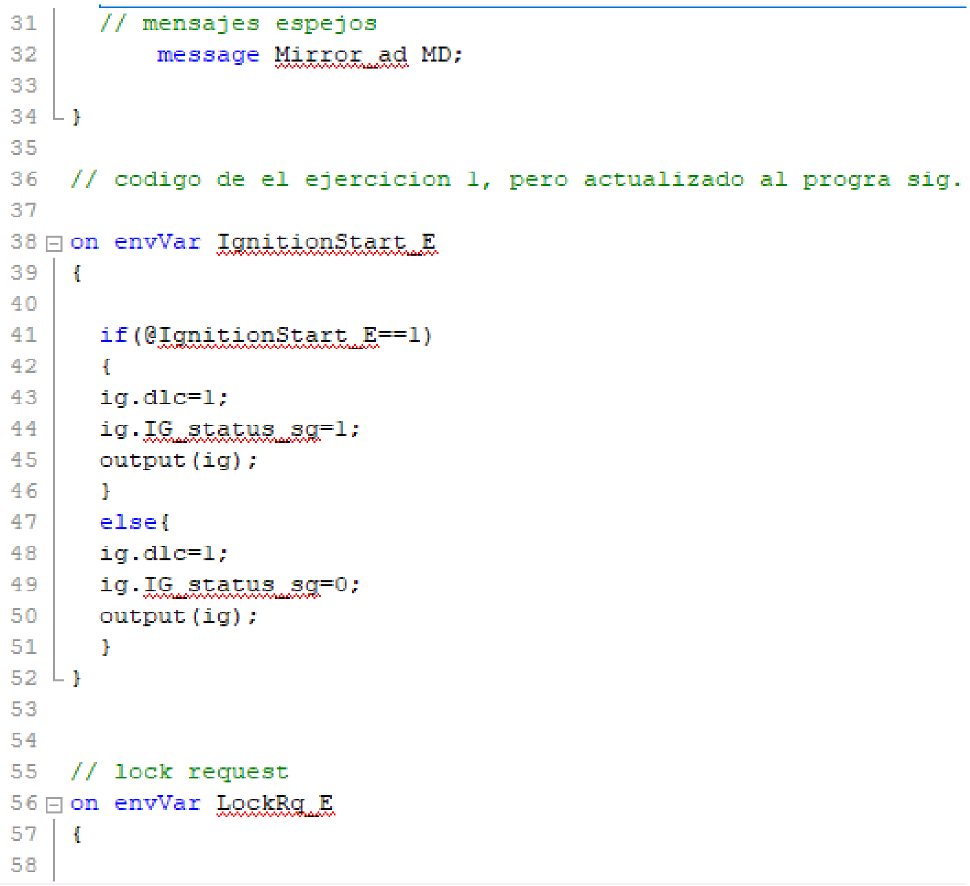
Figure 19. Fragments of the CAPL code for "Door" node.

Figure 20, shows a fragment of the CAPL code (due to the original code being more than 460 lines long) for node “Control” implemented to control the car functions such as the lights, the ignition state, the window lifter, the window movement blocker and the locking/unlocking of doors.

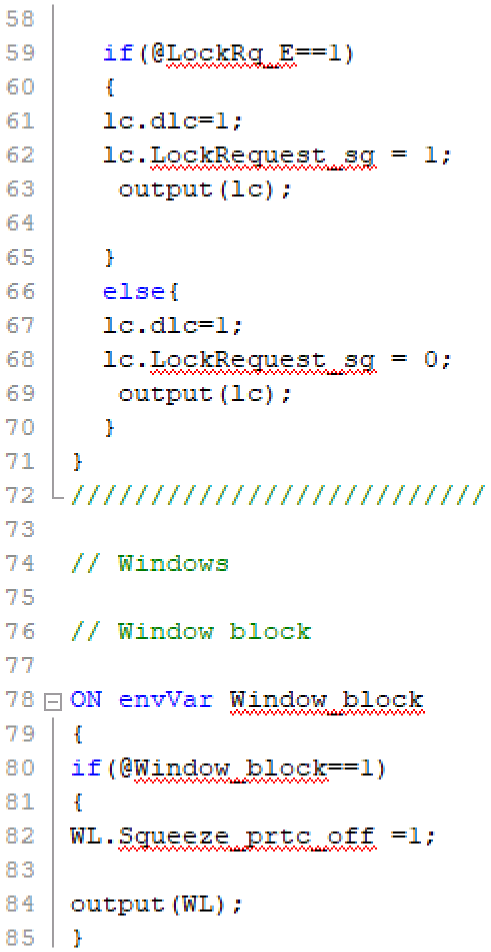


```
/*@!Encoding:1252*/
includes

variables
{
  message Control_Lights CLM;
  msTimer t1;
  msTimer t2;

  // mesages 4 tsk 3

    // mesages 4 tsk 3
  message IgnitionState ig;
  message LockingControlRQ lc;
  message WindowRq Wr;
      msTimer t11;
  msTimer t22;
  msTimer t11R;
  msTimer t22R;
  msTimer t3;
  msTimer t4;
  msTimer t3R;
  msTimer t4R;
  message Window_lifter_sg WL;



  // mensajes espejos
```

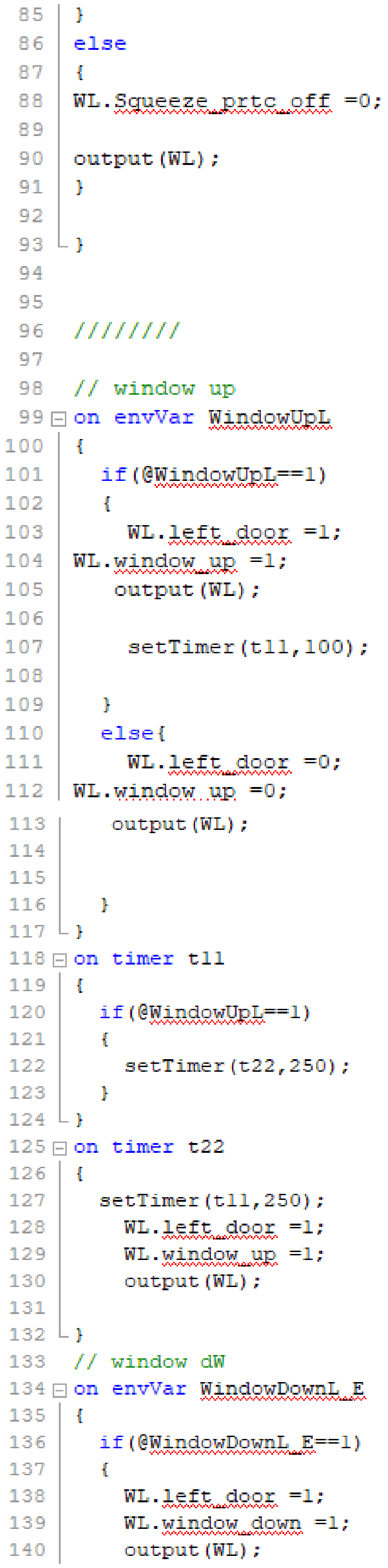
Code continues on next page……



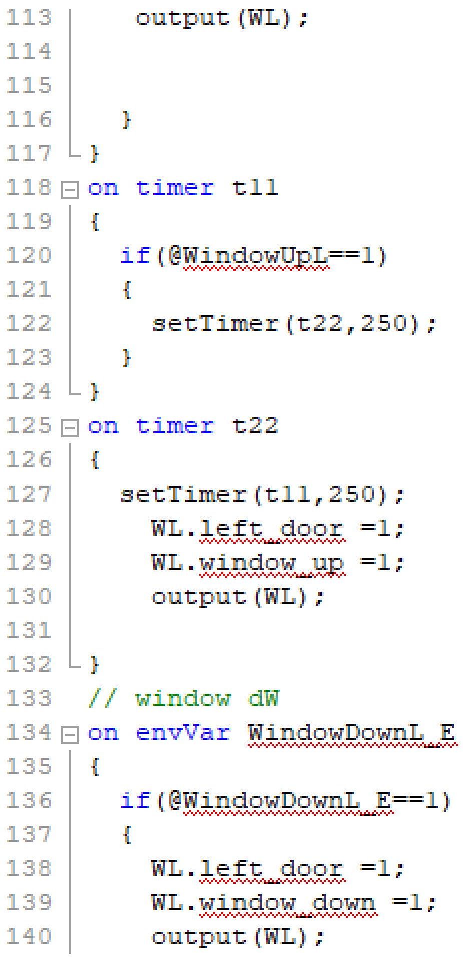
```
  // mensajes espejos
      message Mirror_ad MD;

}

// codigo de el ejercicion 1, pero actualizado al progra sig.

on envVar IgnitionStart_E
{

  if(@IgnitionStart_E==1)
  {
  ig.dlc=1;
  ig.IG_status_sg=1;
  output(ig);
  }
  else{
  ig.dlc=1;
  ig.IG_status_sg=0;
  output(ig);
  }
}


// lock request
on envVar LockRq_E
{

```



```

  if(@LockRq_E==1)
  {
  lc.dlc=1;
  lc.LockRequest_sg = 1;
   output(lc);

  }
  else{
  lc.dlc=1;
  lc.LockRequest_sg = 0;
   output(lc);
  }
}
///////////////////////////

// Windows

// Window block

ON envVar Window_block
{
if(@Window_block==1)
{
WL.Squeeze_prtc_off =1;

output(WL);
}
```



```
}
else
{
WL.Squeeze_prtc_off =0;

output(WL);
}

}


////////

// window up
on envVar WindowUpL
{
  if(@WindowUpL==1)
  {
    WL.left_door =1;
WL.window_up =1;
   output(WL);

    setTimer(t11,100);

  }
  else{
    WL.left_door =0;
WL.window_up =0;
   output(WL);


  }
}
on timer t11
{
  if(@WindowUpL==1)
  {
    setTimer(t22,250);
  }
}
on timer t22
{
  setTimer(t11,250);
    WL.left_door =1;
    WL.window_up =1;
    output(WL);

}
// window dW
on envVar WindowDownL_E
{
  if(@WindowDownL_E==1)
  {
    WL.left_door =1;
    WL.window_down =1;
    output(WL);
```



```
   output(WL);


  }
}
on timer t11
{
  if(@WindowUpL==1)
  {
    setTimer(t22,250);
  }
}
on timer t22
{
  setTimer(t11,250);
    WL.left_door =1;
    WL.window_up =1;
    output(WL);

}
// window dW
on envVar WindowDownL_E
{
  if(@WindowDownL_E==1)
  {
    WL.left_door =1;
    WL.window_down =1;
    output(WL);
```

Figure 20. Fragments of the CAPL code for "Control" node.

Next figures show the functioning of the panel designed to control the indicator system of the car.

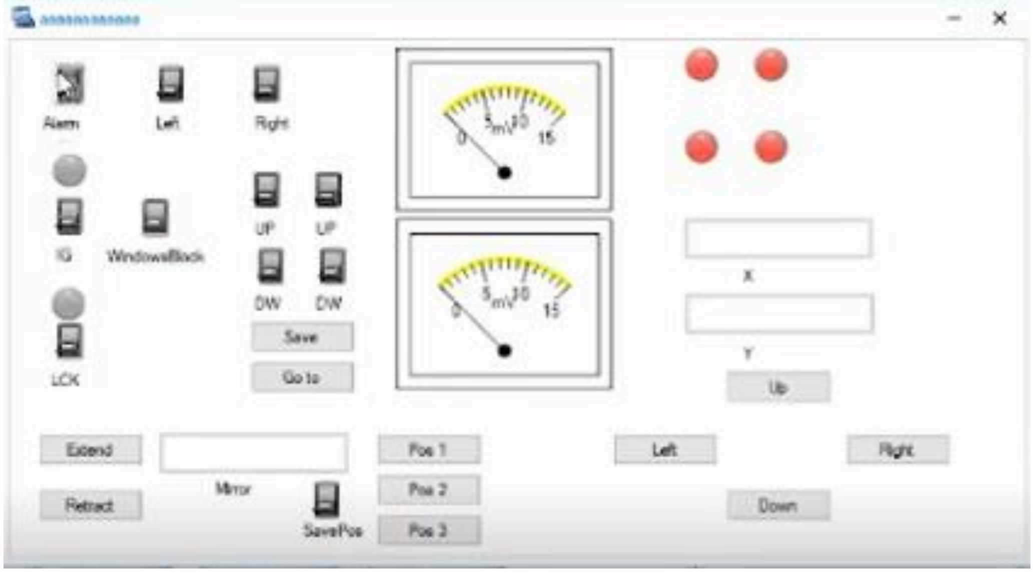


Figure 21. Lamps blinking (red lights) due to the alarm button activated.

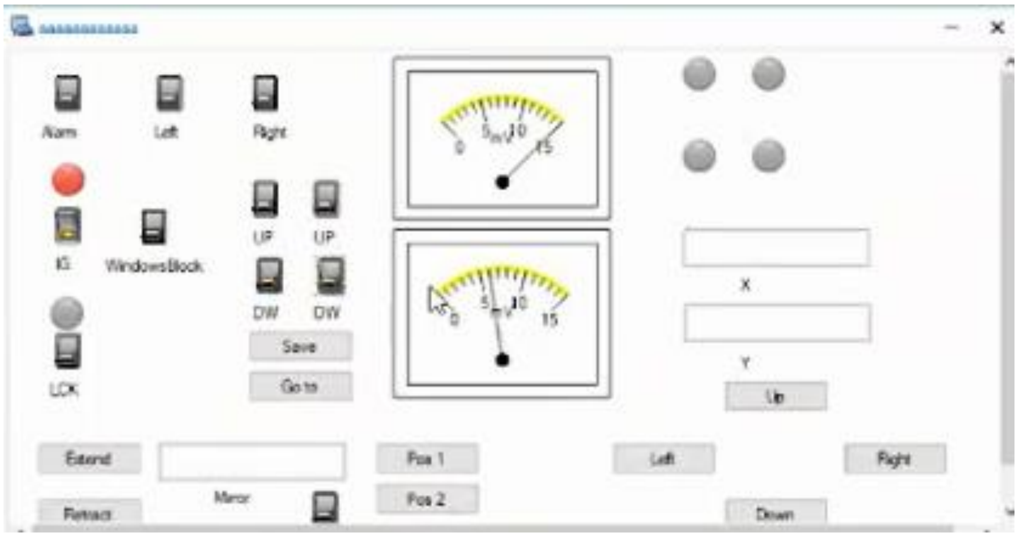


Figure 22. Ignition set to on (red light above "IG" button), left window being completely up and right window moving halfway up (opening percentage represented with needle indicators).

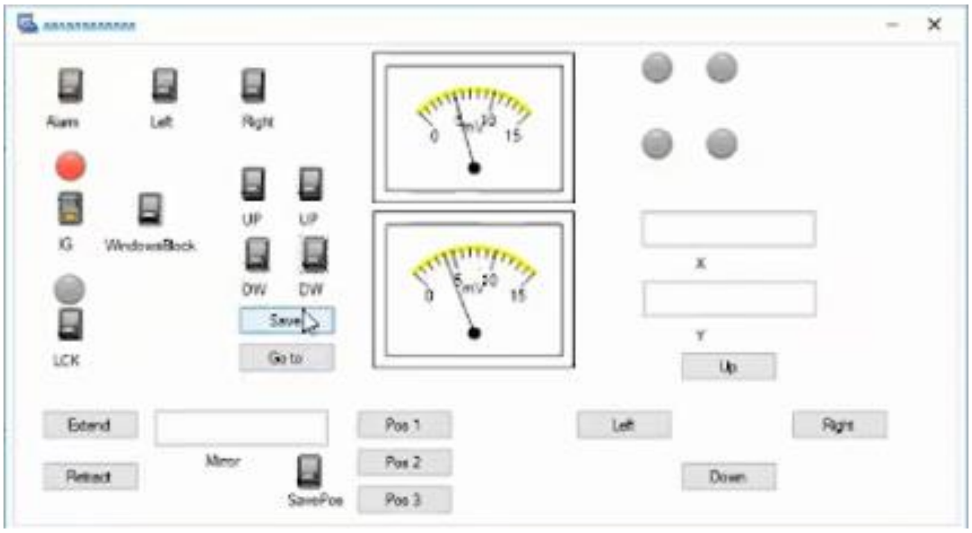


Figure 23. Current position of the windows saved.

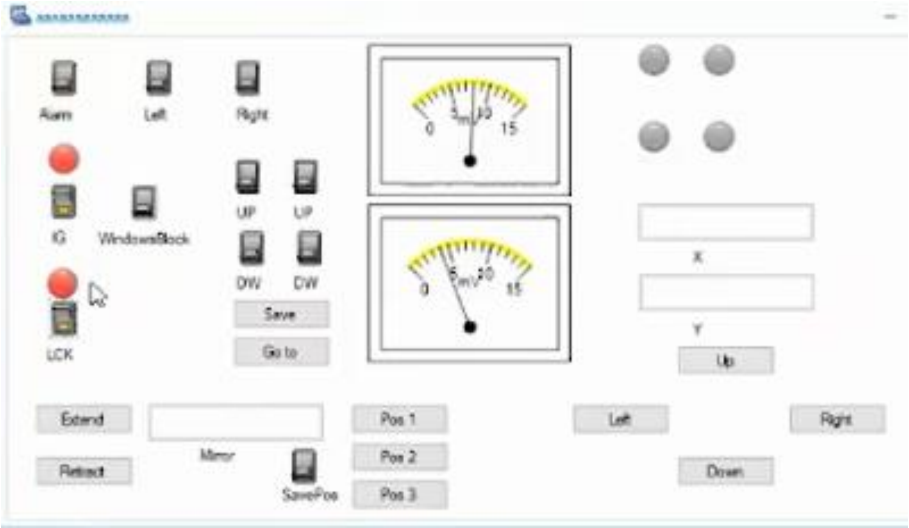


Figure 24. Ignition state set ton on and doors locked (red indicator light above "LCK" button).

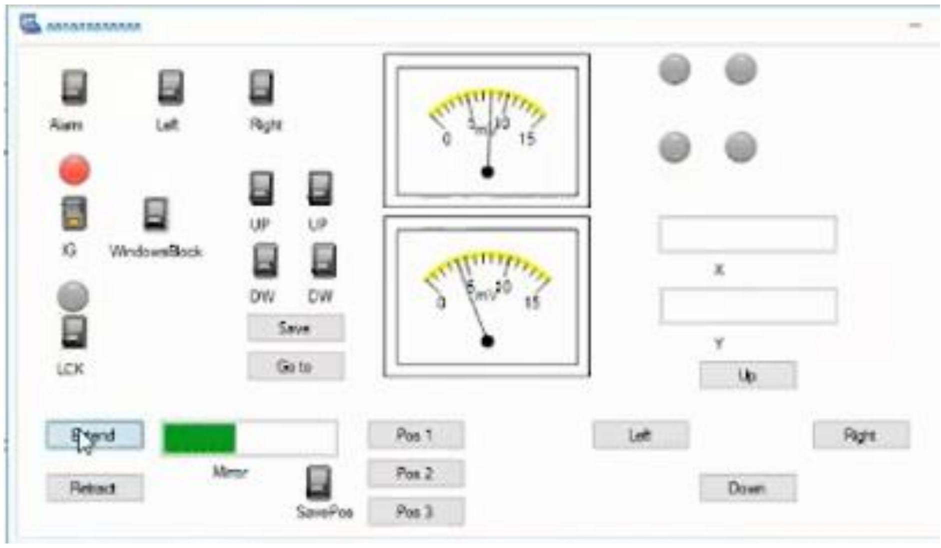


Figure 25. Mirror getting extended almost halfway (represented with green bar indicator).

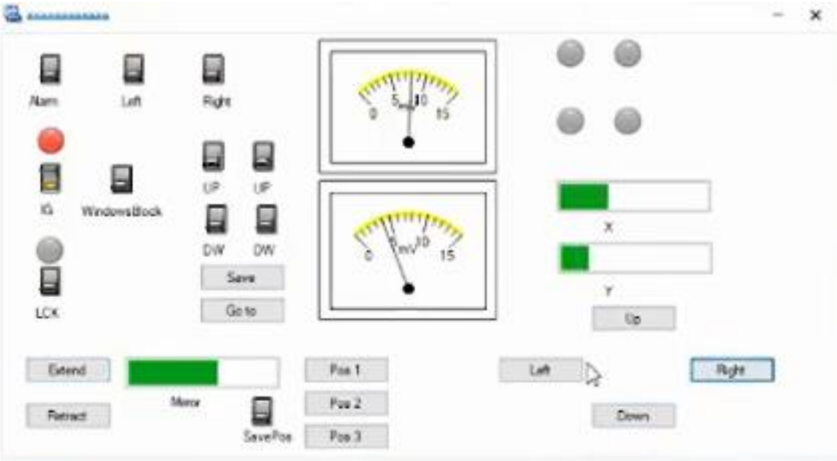


Figure 26. Mirror getting extended a little more than halfway and being moved up and to the right (represented with green bar indicators).

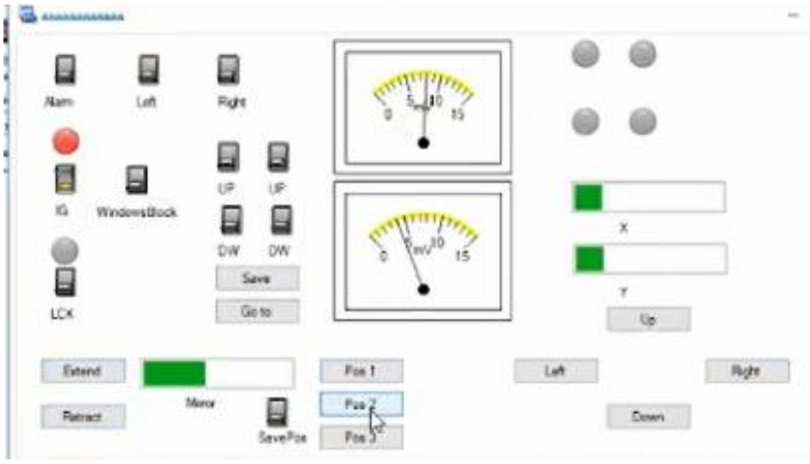


Figure 27. Mirror moving to position 2 (previously saved).

Conclusion

CANoe is such a powerful tool which allowed us to create two complete ECU networks with different nodes each. By making use of our knowledge about CANdb databases, we were able to create the required messages, signals and environmental variables to properly completely the networks. By visualizing the CAN bus activity in the trace window of CANoe, we were able to verify that both ECU networks worked properly by monitoring the sending of messages and signals between nodes.

After completing the tasks and experiments, it was observed how complex just a car's indicating system can be (and in this case, it was only a small system), however, after this work a better idea of how other characteristics of the car could be controlled was developed.

This way, by achieving a successful work for both parts of this work, our knowledge about the CAN Bus, ECU networks, panel designing, CANdb databases and CAPL programming were reinforced and by using all these concepts in a practical application in real life is quite helpful and interesting.

Bibliography

Embeddedc.in. (2015). CAPL Basics. Embeddedc.in. Retrieved April 2, 2022, from: https://www.embeddedc.in/p/n-capl-can-accessprogramming-language-n.html

Vector. (2021). CANoe | ECU & Network Testing. Retrieved March 14, 2022, from Vector Informatik GmbH website: https://www.vector.com/int/en/products/products-a-z/software/canoe/#c216062