MBSD Assignment #4 A.Y. 2024/25

# Purposes

* Integrate the one-pedal controller into a simulated Arduino Uno[[1]](#footnote-1) microcontroller (μC), resorting to SimulIDE[[2]](#footnote-2).
* Interact with the μC through its digital and analog interfaces.

# Instructions

For instructions on how to use SimulIDE and the Platform Support Packages, follow the instructions provided by Prof. Violante in the lectures.

The delivery shall contain:

* The controller Simulink model is used to generate the firmware binary file (plus all the accompanying files needed to make it possible to generate the code again, like .m files containing initializations)
* The firmware binary file to be loaded into the simulated Arduino in SimulIDE
* The SimulIDE project file.
* The PDF or Microsoft Word version of the report.

Is available an example based on a Tank level controller in the folder

The deliverable has to be provided as a . ZIP file up to **June 8th at 23:59. It shall also contain a brief report explaining the** integration process using the following template. It is sufficient that only one of the group members uploads it.

# Model-Based Software Design, A.Y. 2024/25

# Assignment 4 Report

## Components of the working group (max 2 people)

* Federico Bena, 328197
* Michele Lauriola, 332679

# I/O interfaces

*[Please describe the I/O interfaces to interact with the one pedal controller through the electrical interfaces of the Arduino]*

To generate the Arduino Firmware using the Simulink Support Package, the **analog input pins** from to are used for the input interfaces. Each input is converted to double and passed to a gain block to convert the readout in Volts from the 10-bit ADC digital value:

Where is the input value in Volts, is the readout from the 10-bit ADC, is the power supply of the device, that for Arduino Uno is 5 V. After the conversion, each signal is proberly mapped to the range of values needed by the controller:

* *ThrottlePedalPosition1*, *2* and *3* signals are mapped into the range of the normalized position, that is a value between 0 and 1, by means of a gain of 1/5, then the signals are casted to single data type;
* *BrakePedalPressed* is mapped to the range [0, 1] by means of a gain of 1/5, then is rounded and successively converted to boolean data type;
* *AutomaticTransmissionSelectorState* is mapped to its range of integer value between 0 and 4, by using a gain of 4/5, then the value is rounded, casted into data int32 before to be converted into data type enum: TransmissionState;
* for the *VehicleSpeed\_km\_h*, in order to map the range [-60; 240] km/h by means of linear mapping solving the following equations:

Where the **offset k** is setted to that the – 60 km/h value corresponds to 0 V while the **gain m** is computed in order to that the 240 km/h value corresponds to 5 V. The obtained equations for the conversion is implemented in Simulink using gain, constant and adder blocks.

The **digital outputs pins** from to are used at the output interface. The outputs signal exiting from the controller are mapped to 8-bit ADC digital value, by casting to int32 data type, the only expections is for the *TorqueRequest\_Nm* signal that is linearly mapped to the range [-80; 80], before to be converted. The following system of equations is solved:

Where the **offset k** is setted such a way that the digital value 0 corresponds to a value of -80 Nm, while the **gain m** is computed in order that the digital value 255 corresponds to a value of 80 Nm. The obtained equations for the conversion is implemented in Simulink using gain, constant and adder blocks. The the signal is casted to int32 data type.

After conversion is made, a Matlab function is passed to each outputs signal, then a demux driven by the outcomes of the function select the proper outout pin.

The logic used is the following:

* for the *TorqueRequest\_Nm a distinction is made* between positive or null and negative values;
* for the *AutomaticTransmissionState* and the *PedalError* signals, the function only serves to select the right output;
* since the *Warning* signal is implemented by means of an RGB led, the functions allows to activate only the red led for Waning(2) and both red and green leds, resulting in yellow, for Warning(1).

*[Screenshot of the controller model ready for the code generation]*

A diagram of a computer

AI-generated content may be incorrect.

Figure 1: Controller model ready for the code generation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Unit** | **Type[[3]](#footnote-3)** | **Conversion formulas** | **Min[[4]](#footnote-4)** | **Max** |
| *ThrottlePedalPosition1* | - | AI |  | 0 | 1 |
| ThrottlePedalPosition2 | - | AI |  | 0 | 1 |
| ThrottlePedalPosition3 | - | AI |  | 0 | 1 |
| BrakePedalPressed | - | AI |  | 0 | 1 |
| AutomaticTransmissionSelectorState | - | AI |  | 0 | 4 |
| VehicleSpeed\_km\_h | Km/h | AI |  | -60 | 240 |
| TorqueRequest\_Nm | Nm | DO |  | -80 | 80 |
| AutomaticTransmissionState | - | DO | - | 0 | 4 |
| PedalError | - | DO | - | 0 | 2 |
| Warning | - | DO | - | 0 | 2 |

Where is the redout in Volts while **TR** is the value of the torque delivered by the controller.

# Code generation for Arduino

*[Please describe the steps to generate the Arduino firmware using the Simulink Support Package for Arduino Hardware]*

The aim is to perform a full executable algorithm export by using the Embedded Coder app on Simulink The following model settings are chosen:

* Solver: Fixed-step discrete, with step size set to 0.1 s;
* Hardware Implementation: Arduino Uno, with Build action set to Build in order to produce the firmware that will be loaded on a SimulIDE simulation environment, and not on the real board.
* Code Generation: the System target file is ert.tlc, the toolchain is Arduino AVR, set for Faster Runs.
* Code Interface: The code interface packaging was set to Nonreusable function.
* Code Placement: File packaging format set to Modular.

*[screenshot of the controller model instrumented with the blocks of the Support Package for Arduino Hardware]*

*A green square with white squares

AI-generated content may be incorrect.*

Figure : Controller model instrumented with the blocks of the Support Package for Arduino Hardware

# Harness

After the model settings are correctly set, the Build functionality of the Embedded Coder is used to generate the C code. Finally, the firmware file called “controller\_imp.hex” is uploaded on the SimulIDE board for testing (Figure ).

*[screenshot of the harness implemented in SimulIDE]*

A circuit board with many switches and switches

AI-generated content may be incorrect.

Figure : Harness implemented in SimulIDE

## Test stimuli

*[Provide a brief description of the stimuli set chosen to test the controller functionality] Due to the limitations of the SimulIDE, provide stimuli to check the function of the one pedal controller without the whole physical model*

In the first part is intended to test the *AutomaticTransmissionSelectorState* and the *TorqueRequest\_Nm* without inserting any error to the controller: only one of the three potentiometers is connected to not have difference for the three *ThrottlePedalPosition* signal. In the second part the *Warning* signal is tested by inducing errors on the sensors.

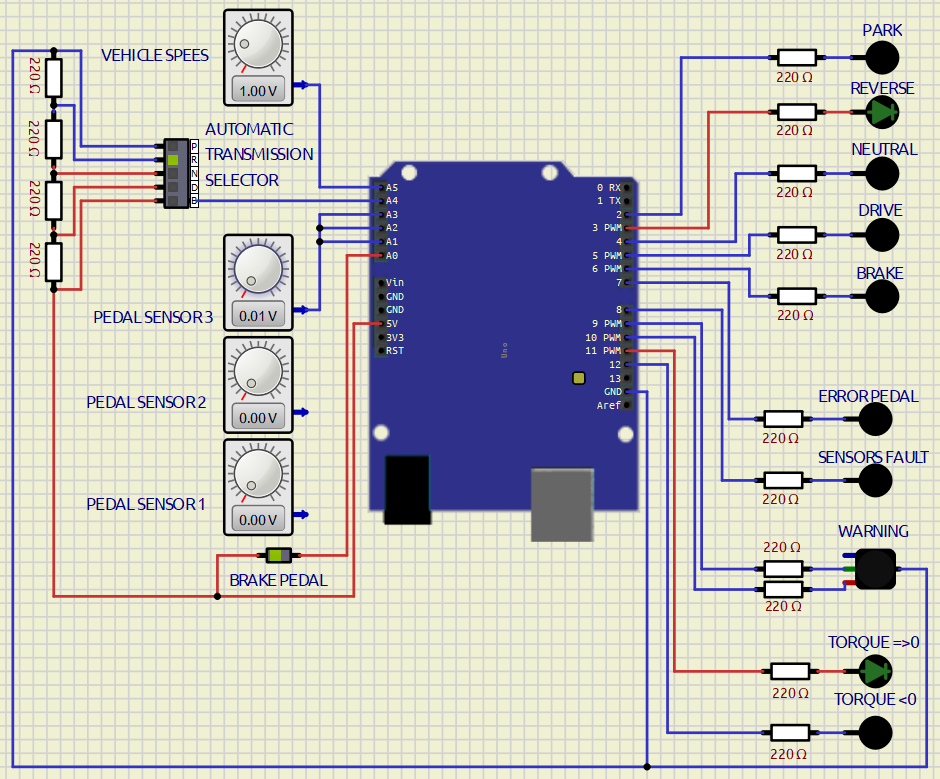
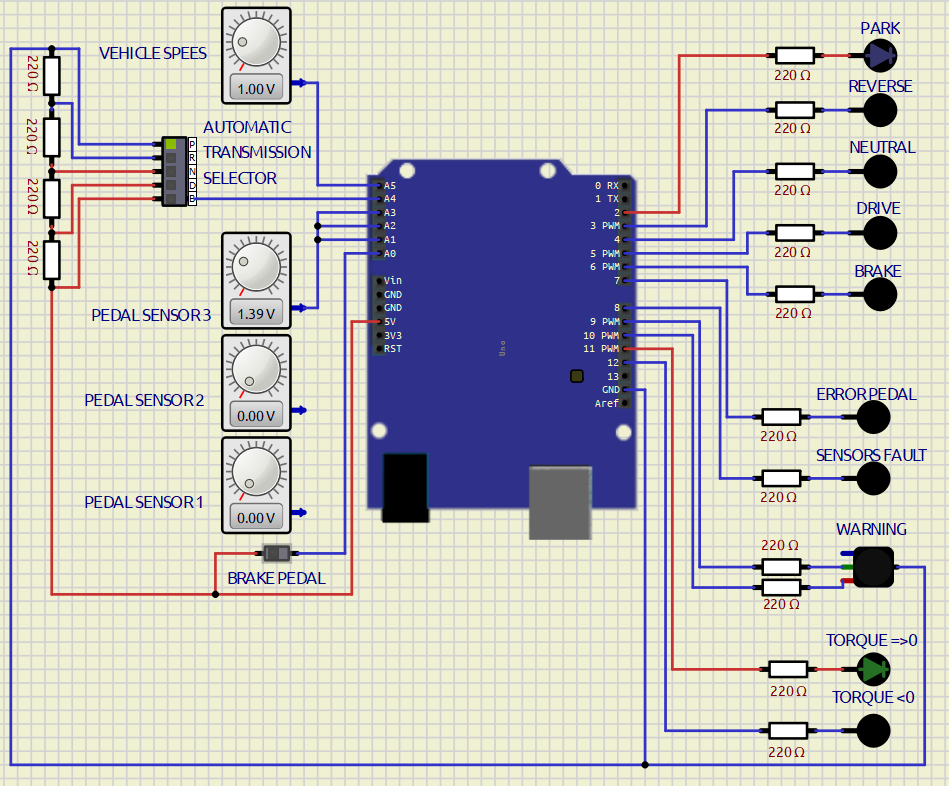


Figure : Transition from Park to Reverse

The simulation is started in Park position, with speed set to 0 km/h, then the selector is set to Reverse, the Brake pedal is pressed to let the controller change the tranmission state. The torque request is null for both states.

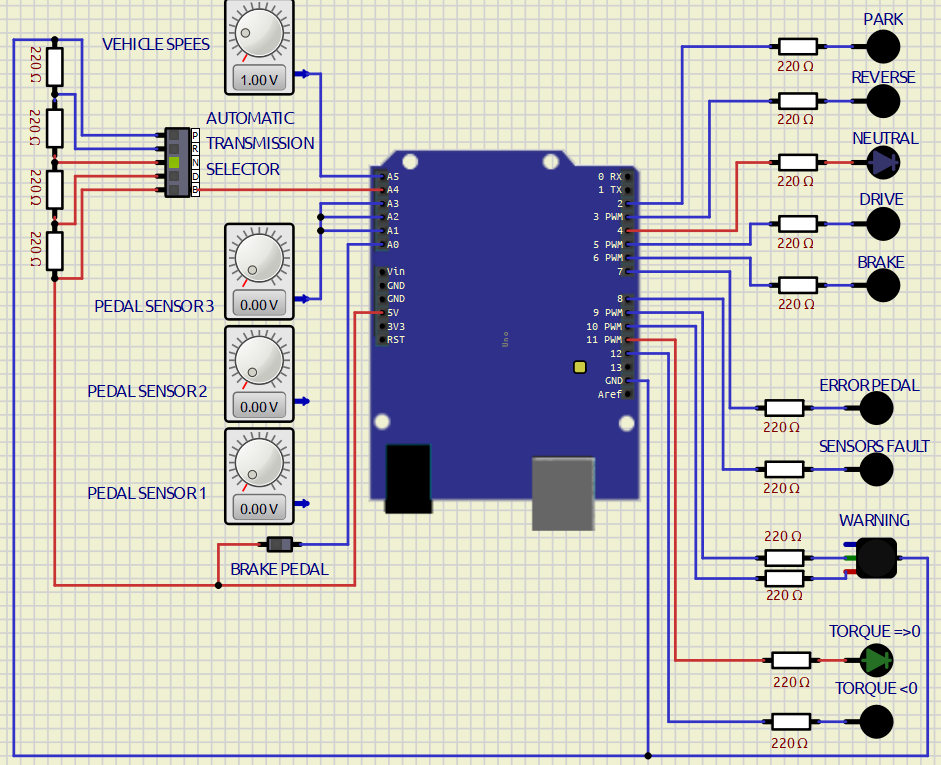
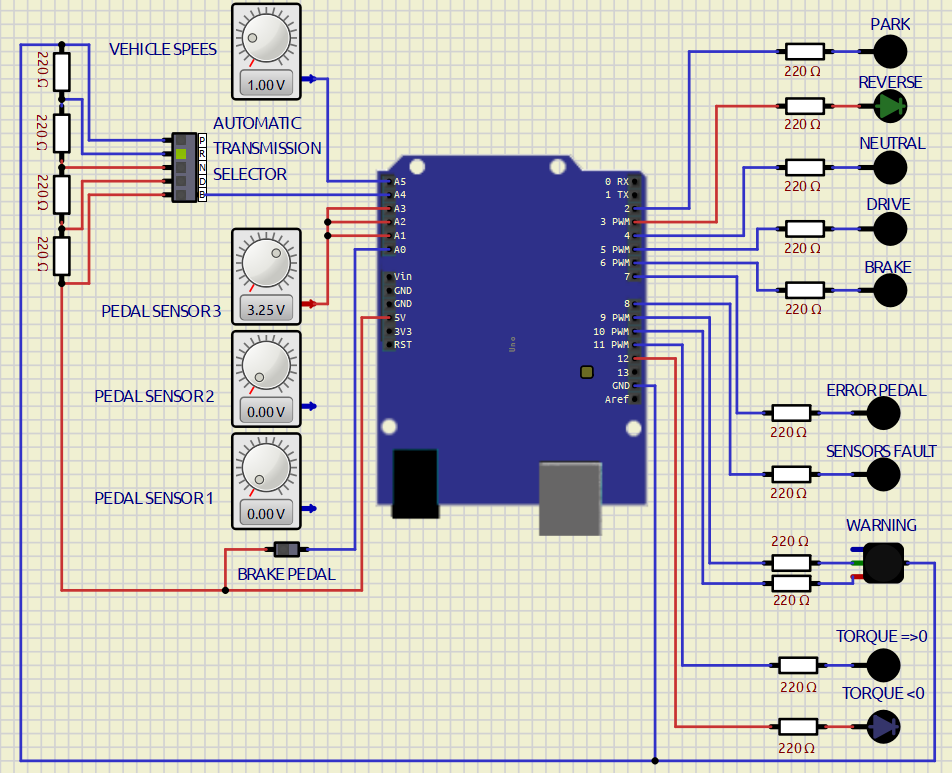


Figure : Transition from Reverse to Neutral

By keeping the selector set to Reverse and by tuning the pedal sensor the torque request results to be negative. Then the selector transits to Neutral, resulting in a torque request equal to zero.

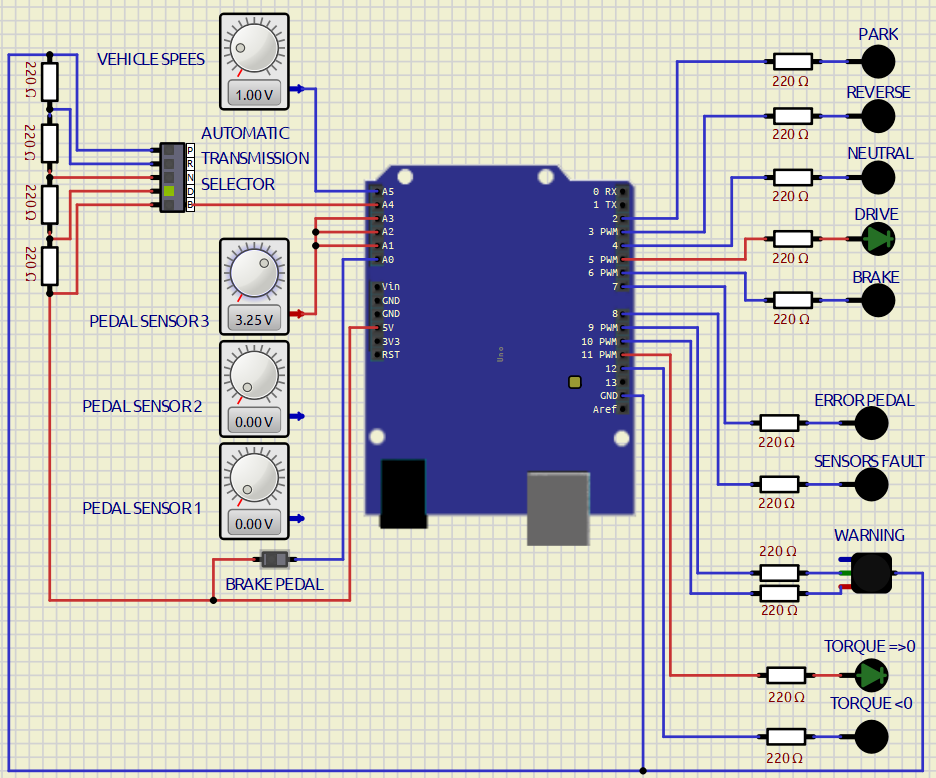
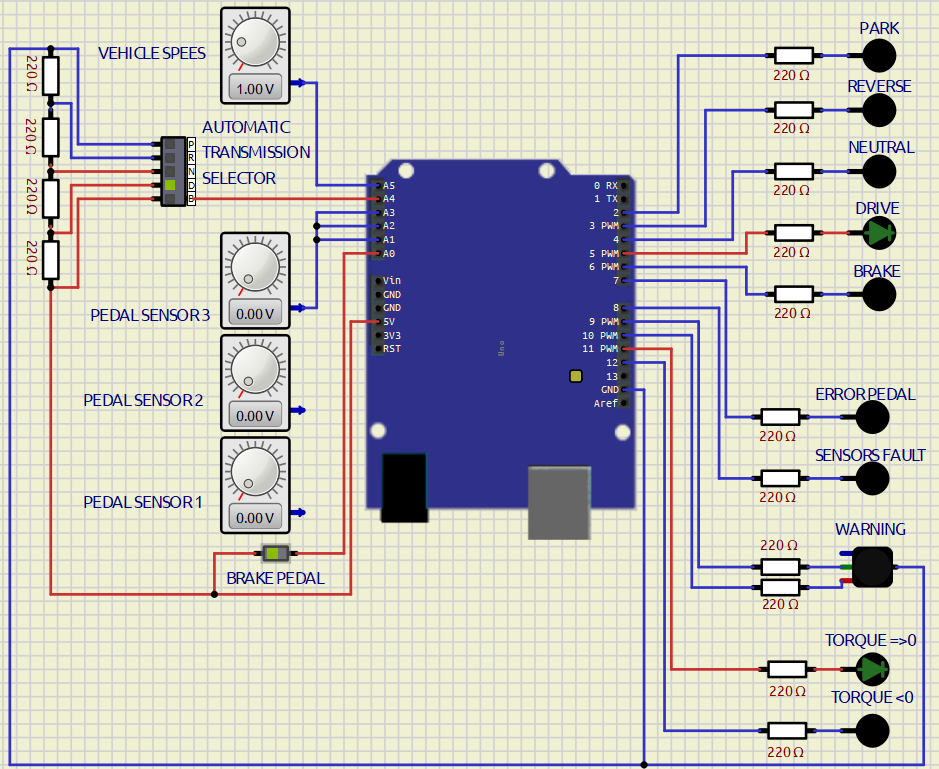


Figure : Transitioning to Drive State

The selector is then set to Drive, the speed must is set to a value under the threshold with the brake pedal that is pressed so that the transition from Neutral to Drive is performed. When the brake is released, it is possible to start accelerating resulting in a positive the torque request.

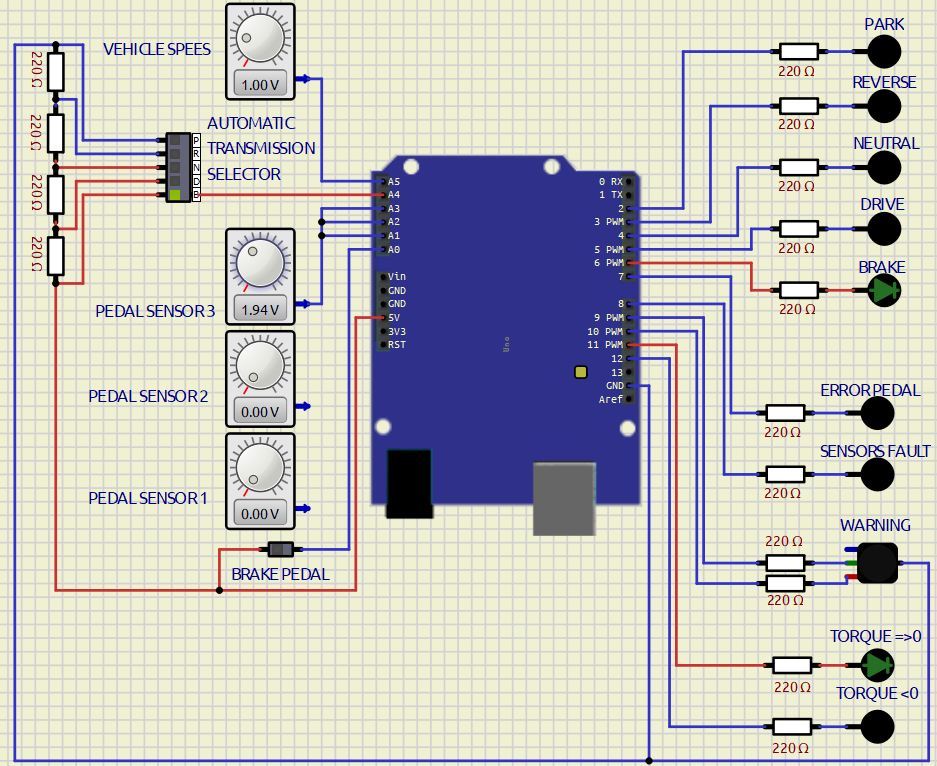
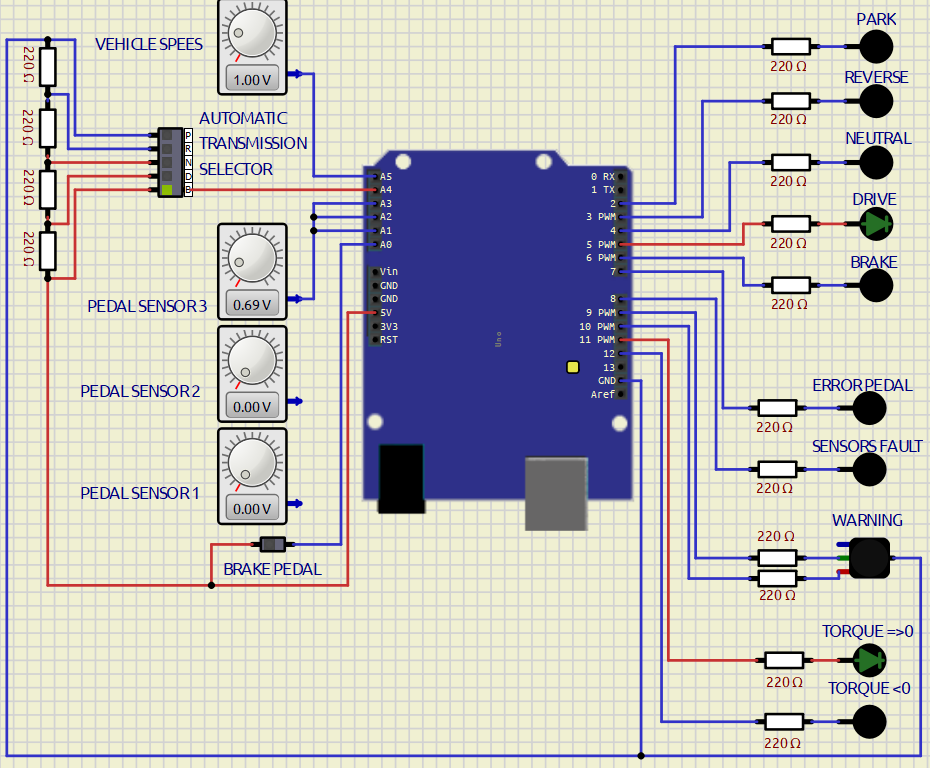


Figure : Transition from Drive to Brake

For the transition to the Brake state, after the selector is set, the transmission will not change until the throttle pedal reaches the threshold of the neutal point.

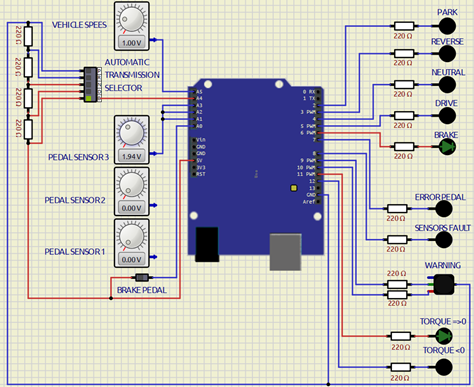


Figure : Brake State

Once the transimission transits to Brake when the throttle pedal is pressed over the threshold the torque request is positive, while below the threshold the torque request is negative.

In the second part of the test all the three sensors of the Throttle pedal are connected individually and the transmission is set to Brake.

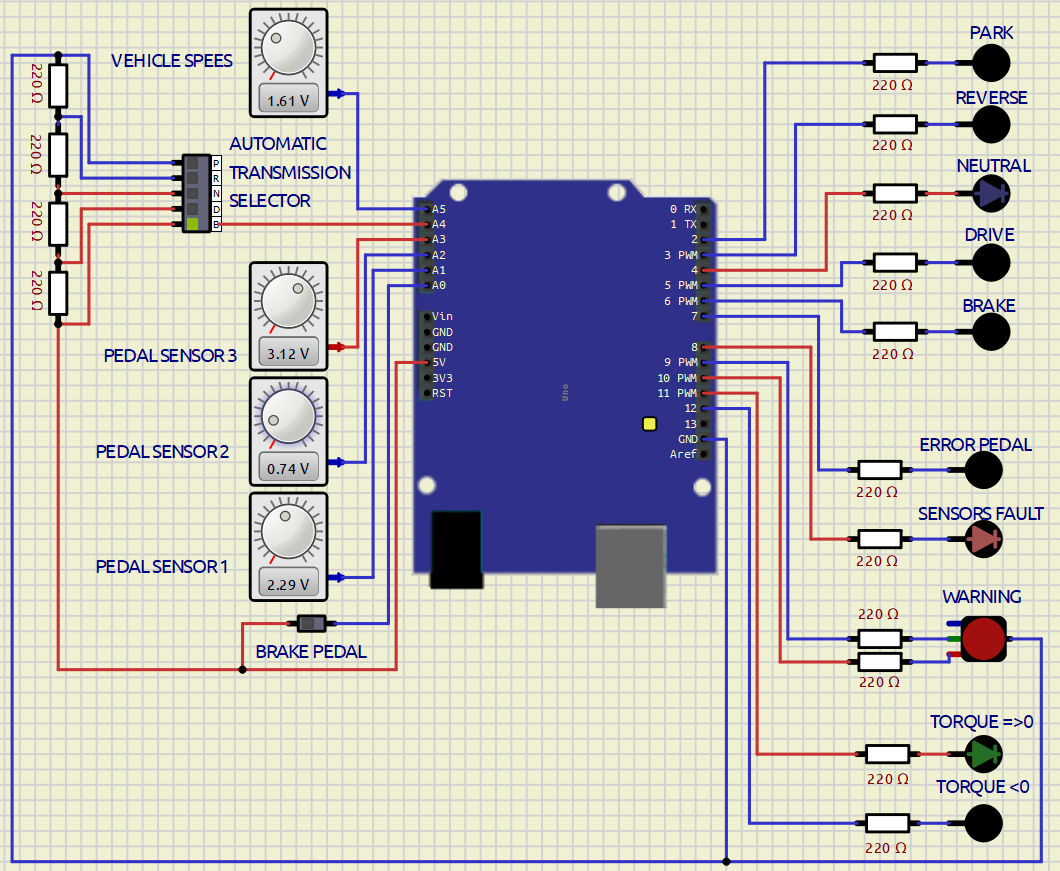


Figure : Sensors fault and Warnings

On the left only one of the three sensors differs from the other two so the Warning is set to level 1 and the pedal error switch on, in this condition everything is still working.

On the right all three sensors differ so the Warning is set to level 2 and the sensors fault switches on, the transmission state tranisits to Neutral and the torque is set to zero.

It is possible to exit from the safe state pressing the brake pedal.

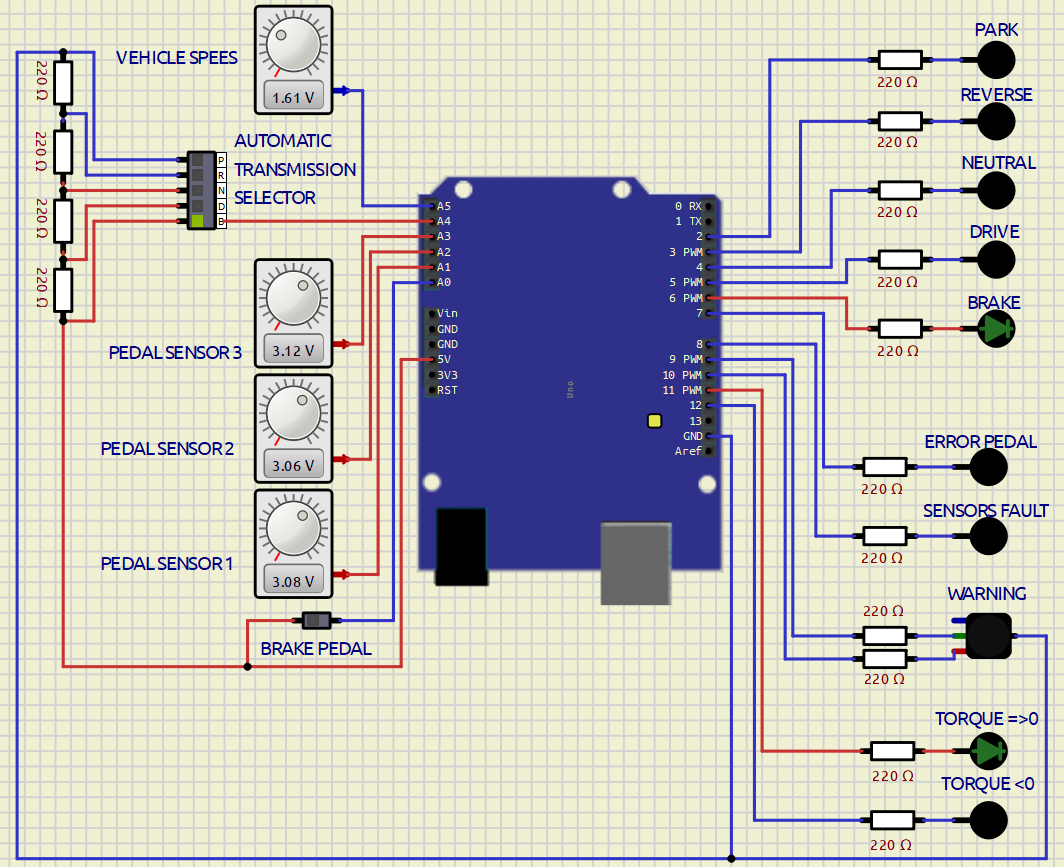
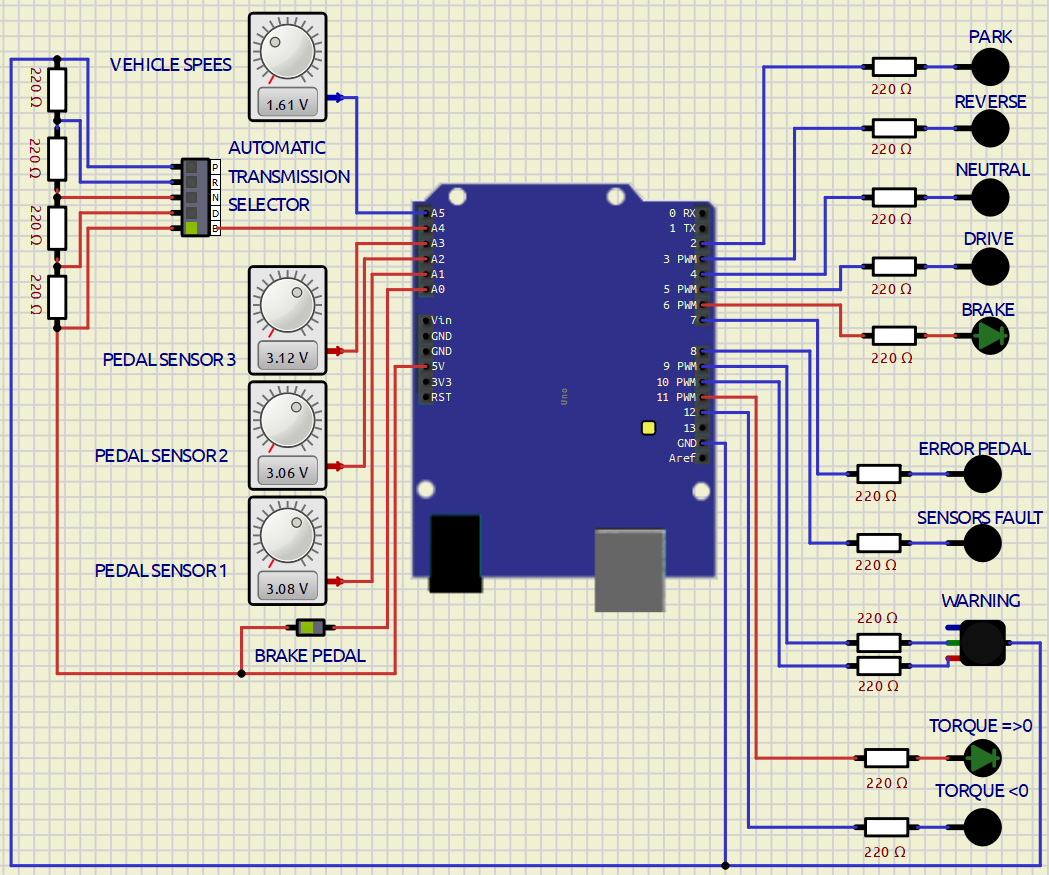


Figure : Exiting the safe state

As the tests performed in the simulation environment returned results confirming the expected ones, the full executable algorithm is loaded on the physical hardware (Arduno Uno) through the Build tool of the Embedded Coder app.

A circuit board with wires and tags

AI-generated content may be incorrect.

Figure 11: Hardware setup on bredboard

The same test performed in SimulIDE are repeated on the hardware mounted on the bredboard (Figure 11), two videos are recorded and uploaded on the delivery folder.

In Video1, the first simulation performed on SimulIDE was played: the compatibility of the torque request with the selected transmission mode was tested.

In Video2, the second simulation performed on SimulIDE was played: the response of the Warning to the introduction of throttle pedal sensor errors was tested.

1. Arduino Uno Board Anatomy <https://docs.arduino.cc/tutorials/uno-rev3/BoardAnatomy>, last visited on 14/05/2025. [↑](#footnote-ref-1)
2. SimulIDE, https://www.simulide.com/p/home.html, last visited on 14/05/2025. [↑](#footnote-ref-2)
3. Digital Input (DI), Digital Output (DO), Analog Input (AI).

   For AIs, provide the conversion formula from input voltage to the measurement unit data (indicating also how to perform the conversion from the raw reading of the ADC). [↑](#footnote-ref-3)
4. The Min/Max values that can be handled due to the conversion formula shall be expressed in the measurement unit specified in the Unit column. [↑](#footnote-ref-4)