**Sensor array report**

**Content**

[1. Introduction 1](#_Toc566418062)

[1.1 Choice of sensors and CAN shield 1](#_Toc110657510)

[1.2 Choice of microcontroller 2](#_Toc366687673)

[2. Coding the Raspberry Pi Pico 3](#_Toc1165885880)

[2.1 Introduction into the Raspberry Pi Pico SDK 3](#_Toc424866518)

[2.2 Communication with the sensors and the CAN shield 3](#_Toc1075033290)

[2.3 Initialization of the accelerometer 4](#_Toc1308733200)

[2.4 Code 4](#_Toc1401167613)

[3. Let’s get physical 9](#_Toc270802948)

[3.1 Schematics 9](#_Toc731989193)

# **Introduction**

The rear sensor array is the part of the car that is formed of all the sensors in the rear part of the vehicle as well as a microcontroller that reads the data from them. It communicates via CAN with the CCU (central computing unit). The microcontroller will send the data from the sensors to the CCU.

## 1.1 Choice of sensors and CAN shield

The sensors that we are using are the MPU6050 accelerometer that measures the acceleration on all three axes. This collection of data will be correlated with the data of the accelerometer in the FSA (frontal sensor array) in order to see the differences of acceleration between the frontal and rear part of the vehicle. In this way, we can detect skids as well as differences in downforce created by the aerodynamic packet.

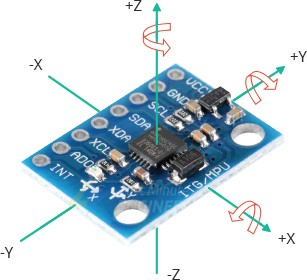


Figure 1 MPU6050 acceleromter board

We are also using another sensor for the second sensor array. It is called I2C-ACC-8700 from gravitech. It also measures the acceleration on all three axis and also the intesity of the magnetic field. By measuring the intesity of the magnetic field we try to eliminate the influence of the gravitational acceleration.



Figure 2 I2C-ACC-8700 accelerometer + magnetometer board

We are also using potentiometers that measure the load on the suspension. Since we are using 2 coil overs in the rear part of the car, the RSA will measure those two potentiometers mounted on the coil overs. The potentiometers act as travel sensors for the suspension.



Figure 3 Shock absorber travel sensor

~~It order to communicate with the CCU, the RSA must have a CAN module in order to be able to transmit data. The microcontroller that we are using does not have integrated CAN support, therefore we chose the MCP2515 as a CAN shield.~~

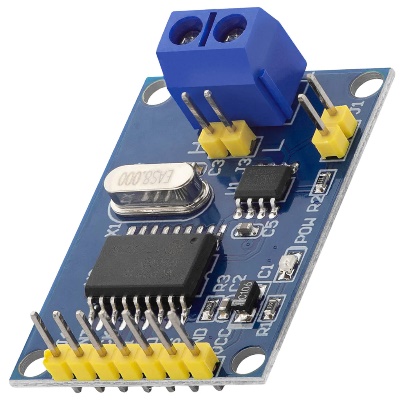


Figure 4 MCP2515 CAN shield

In order to communicate on the CAN bus we used the PIO’s of the Raspberry Pi Pico in order to emulate a CAN module. The only additional part was the CAN transciever who transform the CAN\_RX and CAN\_TX signals into CAN\_L and CAN\_H signals. The code that we used is available at the github page: <https://github.com/KevinOConnor/can2040>.

DISCLAIMER: IN ORDER FOR THE CAN TRANSMISSION TO WORK, THE TRANSCIEVERS MUST BE CONNECTED AT THE SAME GROUND VOLTAGE LEVEL.

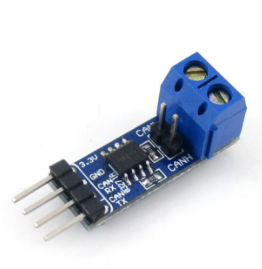


Figure 5 Waveshare CAN transciever SN65HVD230

1.2 Choice of microcontroller

For the brains of this RSA we went with Raspberry Pi Pico, because it has a very low price and it is powerful enough for our application. The community offers a lot of documentation about everything. Even though the SDK is pretty hard to set up at first, the programming learning curve is pretty good. It also has the advantage of boasting an ARM microprocessor which is powerful enough to sustain an RTOS which brings the advantage of utilizing the microprocessor at it’s maximum capacity, without introducing unecessary delays, as in the case of the classical programming.

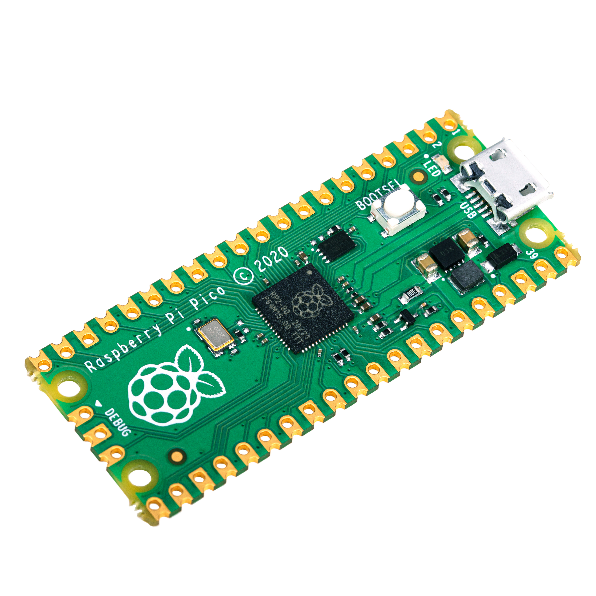


Figure 4 Raspberry Pi Pico microcontroller

# **Coding the Raspberry Pi Pico**

## 2.1 Introduction into the Raspberry Pi Pico SDK

The Raspberry Pi Pico does not have a IDE as you probably expect it have. However, Visual Code Studio can be set up to use the SDK for Pico in order to behave as the IDE for Arduino, for example. In order to get everything set-up, please access the link here: <https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf>. Beware of the fact that the linux script is made for Raspberry boards that run some kind of modified Debian and it won’t work on Debian based distros as Ubuntu (been there done that).

## 2.2 Communication with the sensors and the CAN shield

The Raspberry Pi Pico has i2c as well as spi support. In order to control the either of the accelerometers, it will communicate with it via i2c. ~~The CAN shield is controlled via spi.~~ The potentiometers will be directly connected to some GPIOs (general purpose pins) that can measure the voltage they output. In order to determinate the position of the travel sensor, the formula from below is used:

position = Vmeasured / Vsupplied \* 100 [percentage]

## 2.3 Initialization of the accelerometer modules

Both accelerometers have an initialization function that will reset their previous settings and make sure that they will be behaving correctly from that moment. A first point of interest is the resolution of their acceleration measurement. In order to correlate the accelerations, we will set the same resolution of 4gs on both accelerometers.

For the MPU6050 I implemented a way of reading the gyroscope in order to obtain the speed in degrees/s. This can be useful in a later implementation of a variable that retains the position of the car.

For the I2C-ACC-8700, besides the acceleration measurement, I also implemented the measurement of the magnetic field.

## 2.4 Code

**3. Let’s get physical**

The physical implementation of this project will contain the schematic of the board and the PCB design. It will not contain the case in which the electronics will reside. The schematics and pcb designs were done in KiCAD.

## 3.1 Schematics

I made the decision to not implement each of the presented modules on the PCB, however I will place appropriate jumpers in order to make a connection between my PCB and the standalone modules of CAN (MCP2515) and Raspberry Pi Pico. Besides them, there will be RJ45 connectors for the CAN bus and the power rails.