Autonomous line-following car for Mercedes Benz

Intro

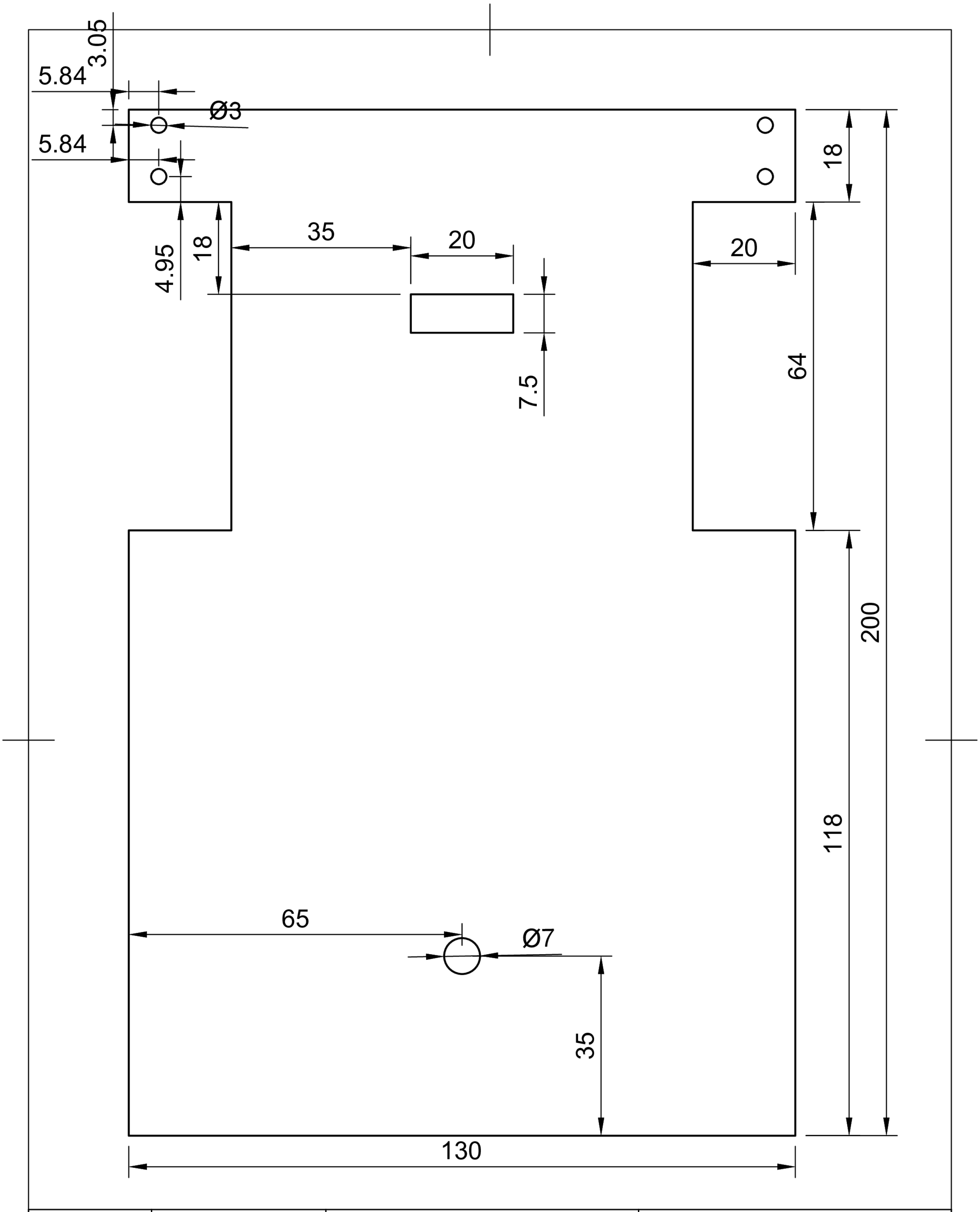
In this project I will make an autonomous line-tracking car. The car is a prototype for the Mercedes Benz museum. Once done, the car will be able to perform the following actions:

1. Follow a white line
2. Detect obstacles in it's way using a HC-SR04 sensor
3. Detect if it has collided with anything
4. Show the battery voltage through a RGB-led indicator
5. Show the status of the vehicle through 4 different leds
6. It'll have a startup-tune
7. Web interface (grafana)
8. It'll also be able to stop at a traversal line at which point it either starts driving after 5sec or after the pressing of a button

Components

1. ESP-32
2. 1 5xTCRT5000 IR sensor with Limit Switch
3. 2 5V Motor
4. 1 L293d Motor Driver
5. 1 LM 1117 T5,0 Voltage Regulator
6. 2 9V Batteries
7. 1 Switch
8. 1 DC-female jack
9. 1 RGB-led
10. 1 MCP23016 I/O Expander
11. 1 HC-SR04 Ultrasone Sensor
12. 4 leds (green, orange, red, blue)
13. 1 Button
14. 1 Buzzer
15. 1 Castor Wheel
16. Wires

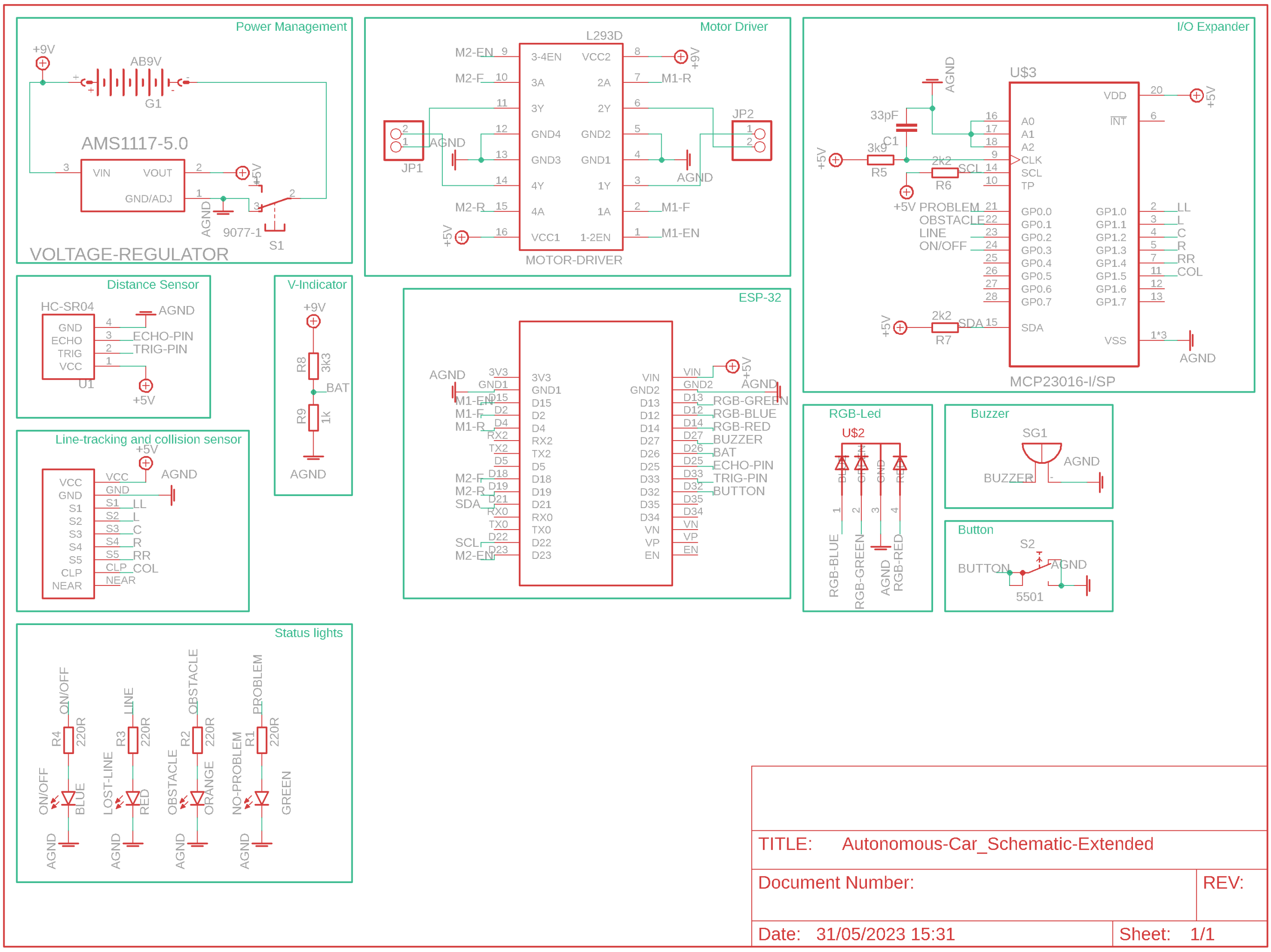
Body Plate



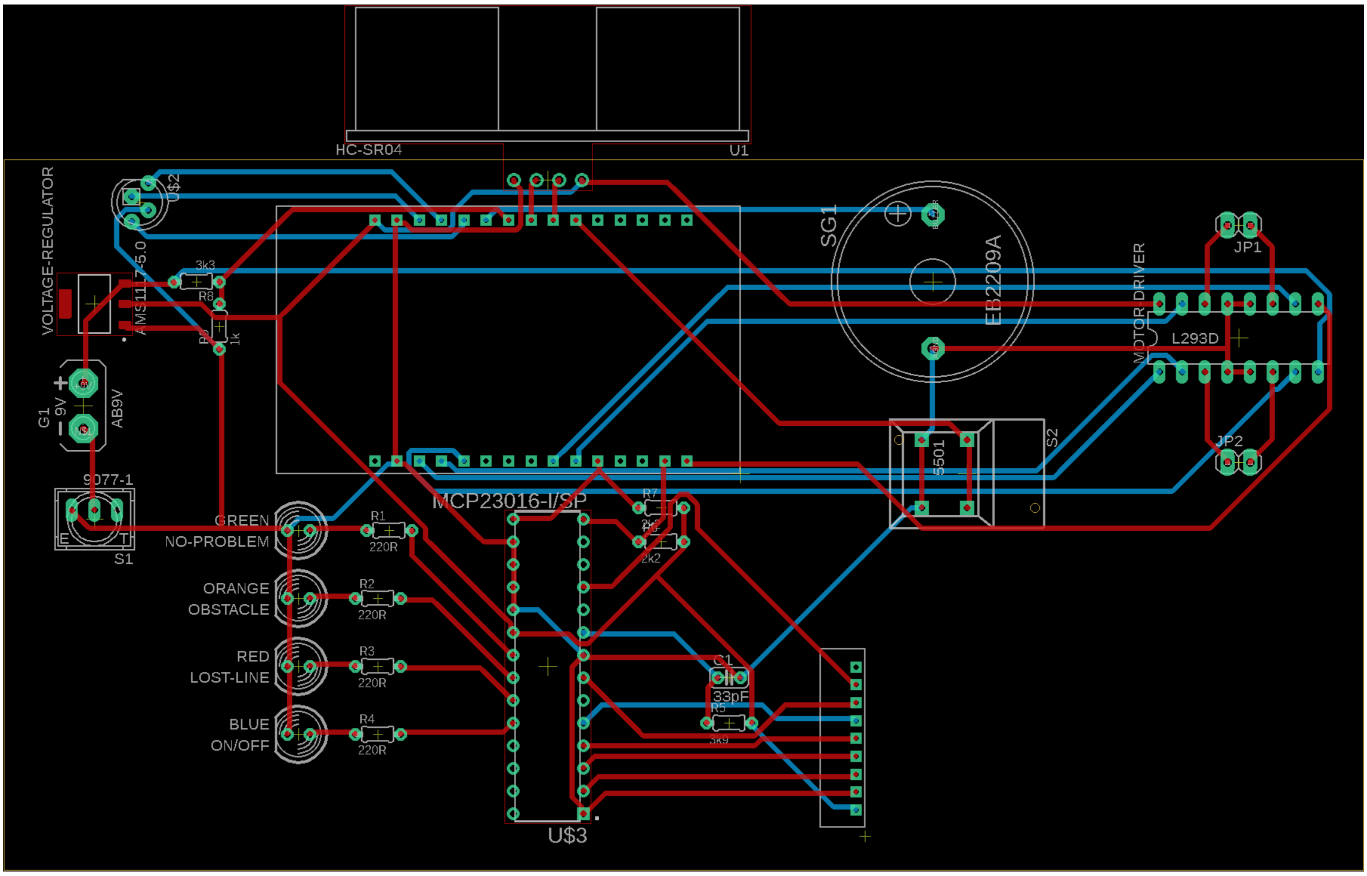
Dept.	Technical reference	Created by Nick Bosschaert 30/04/2023	Approved by
		Document type	Document status
		Title Autonomous line tracking car Design	DWG No.
			Rev. Date of issue Sheet 1/1

The Schematic & PCB

The Schematic (.sch eagle file) can be found in the ./PCB&Schematic/ Folder



An Image of the PCB and the gerber files can also be found there.



The Code

Without MQTT

The full .ino codefile can be found under ./Arduino/Final.
This code has a couple different elements/sensors and I will be explaining the full code per element.

MCP23016 I/O Expander

Pre-Setup

In the pre-setup we are including the library that is necesary to use the MCP23016. This code also changes the call of the MCP23016 to just MCP which saves space on the microcontroller and makes writing the code easier. We also define the port A (0.0-0.7) and port B (0.1-0.7) data register addresses so that the sketch can interact with both registers. And at last we declare some initial variables of the library that it will need.

```
//------MCP23016------
#include <MCP23016.h>

MCP23016 MCP;

#define GPA 0x00  //port A data register address
#define GPB 0x01  //port B data register address

uint8_t mcpA;  //internal variable of the library - required to be declared as is
```

```
uint8_t mcpB;  //internal variable of the library - required to be declared as is
```

Setup

In the setup we initialise the expander library with it's I2C address (0x20) and we also initialise Wire for the I2C connection. We also define the pinmodes of all the connected sensors (INPUT) and leds (OUTPUT).

```
  //------MCP23016 I/O Expander------
  Wire.begin();
  Wire.setClock(100000);

  MCP.init(0x20);

  MCP.pinMode(lineSensorLL, INPUT);
  MCP.pinMode(lineSensorL, INPUT);
  MCP.pinMode(lineSensorC, INPUT);
  MCP.pinMode(lineSensorR, INPUT);
  MCP.pinMode(lineSensorRR, INPUT);

  MCP.pinMode(collisionSensor, INPUT);

  MCP.pinMode(problem, OUTPUT);
  MCP.pinMode(obstacle, OUTPUT);
  MCP.pinMode(line, OUTPUT);
  MCP.pinMode(onOFF, OUTPUT);
```

Loop

Inside of the loop we use the MCP mainly for getting sensor data and changing the status leds state. The majority of this will be explained later in this document but I'll give some examples.

To put a led on we do:
`MCP.digitalWrite(onOFF, HIGH);`
Here `onOFF` is the led from which we want to change the state and `HIGH` means that we want to put the led on. If we would want to put the led off, then we would use `LOW` here.

To read a sensor's value we would do:
`MCP.digitalRead(sensor1);`
In which `sensor1` is the sensor we want to read the value from.

MCP23016 I/O Expander

Pre-Setup

Setup

Loop

With MQTT

The full .ino codefile can be found under ./Arduino/Final-MQTT.