ARMobile Documentation

2021.05.21

Developers

- Csókás Bence
- Koloszár Gergely

Specification

Te main goal was to create a simple car model, that can be controlled remotely, without any wires.

Hadware:

The model can be separated into 3 layers:

- stm32 based nucleo board
- drive electronics
- base board, and mechanics

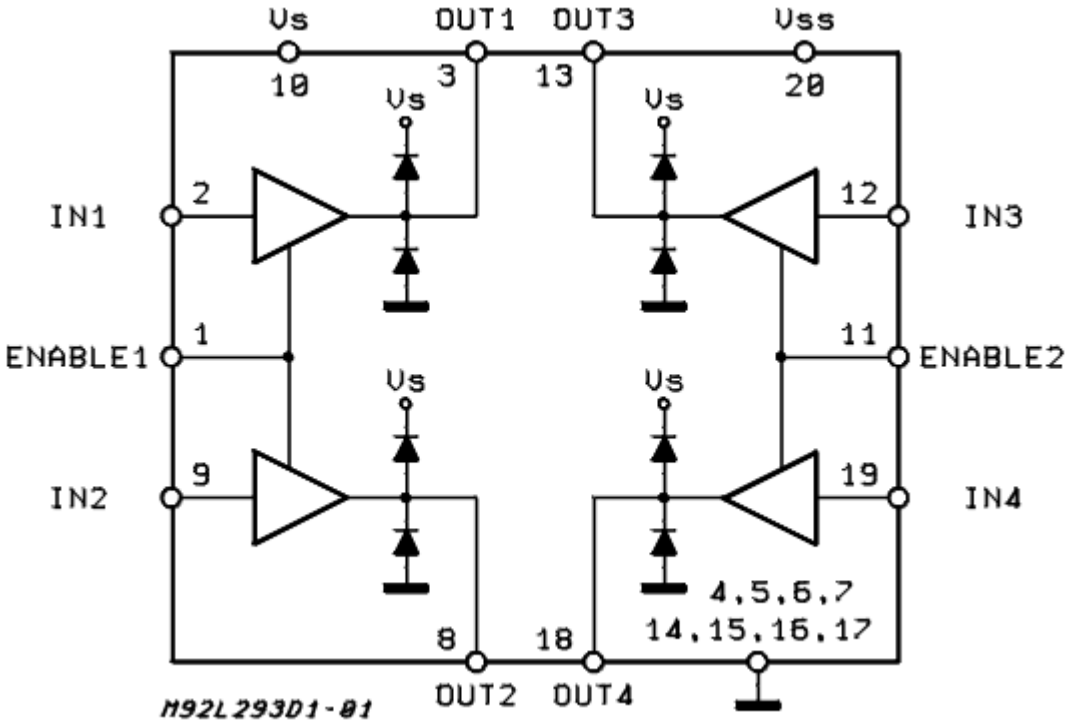
controller

The main controller is chosen to be an `STM32F303RE` controller. For a first project, this controller has enough computing capacity to handle any task that might appear, and leaves room for improvement. The board type is a `nucleo-64` board, which is small enough to fit for a project like this, but still comfortable to work with.

drive electronics

Summarizing the required capabilities the optimal choice seemed to be two `L293D` circuits. These are simple motor drive ICs containing two modules each. Each module has three input, and two output, pins with the following functions:

- output 1 and 2 are connected to the motors pins
- input 1, and 2 are amplified to create outputs, therefore they control rotation direction.
- enable input simply enables the two output amplifiers, and makes it easier to drive the circuit with PWM signals



Each IC contains two of these modules, as the picture shows therefore two ICs are required, to drive the four motors. To ensure that the motors on one side always rotate in the same direction, the appropriate input pins and enable pins are connected in hardware.

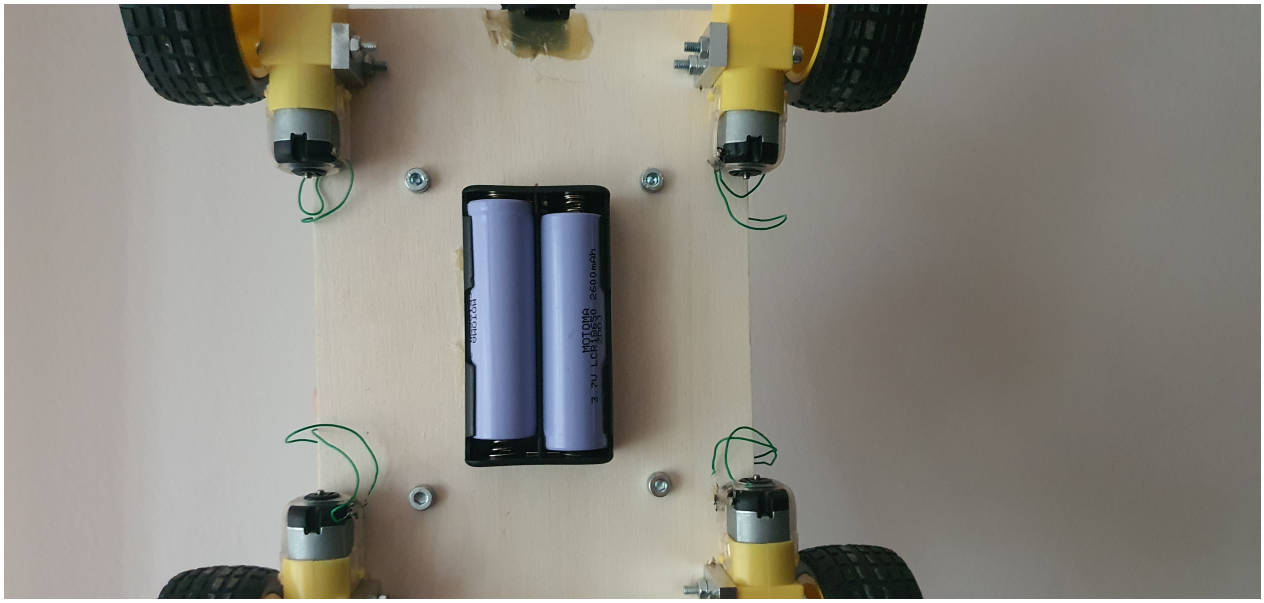
The two ICs, pin arrays for the nucleo and the wiring are mounted on test-PCB, which functions as the base for the electronics. The whole PCB is held in place by four screws, and mounted on a wooden plane.

To have a wireless communication with the board, a `HC-05` module was also placed onto the PCB. This small module have given a bluetooth connectivity option, by having an RX and TX pins and converting bluetooth messages into standard `8N1 UART` signals. This module is used to control the vehicle.

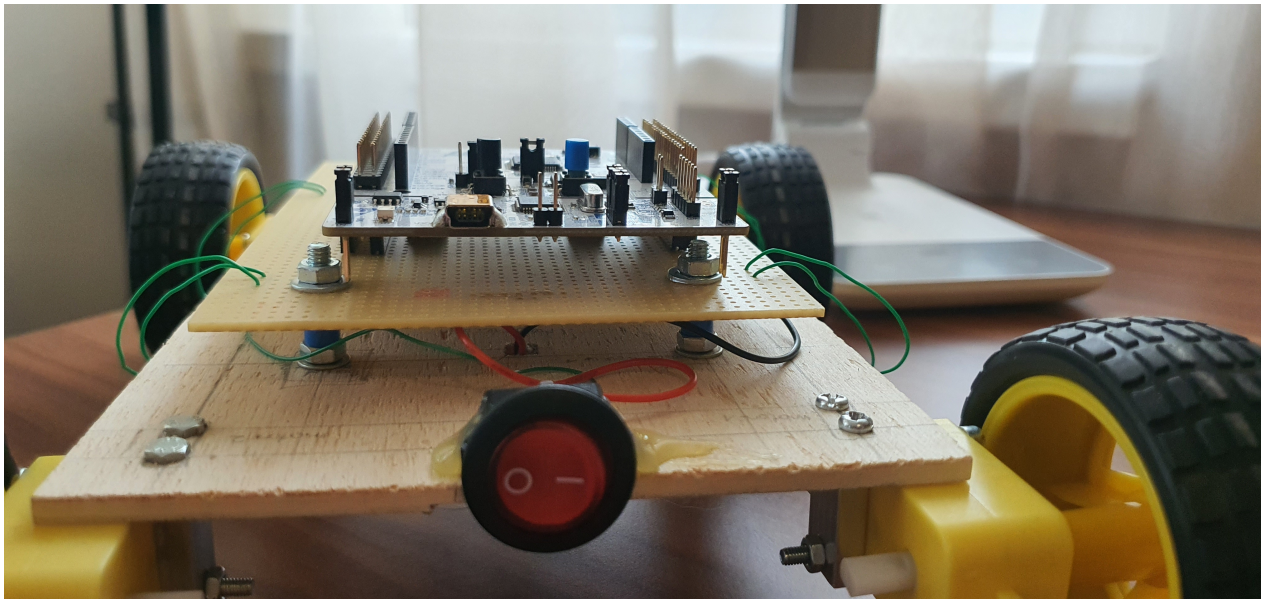
Mechanical layer

The mechanical base for this project was a small wooden plane. Four small DC motors (RM-17) are mounted onto this plain, and it connects to the main PCB with four screws. The main power source is located on the bottom of this plain. To have enough driving voltage, two serial 18650 Li-Ion cells, were installed on board. They provide 7.2 V for the motors, and are used as the main source of power for the controller as well. A simple power switch is installed onto the back of the vehicle, which can remove power entirely from the system.

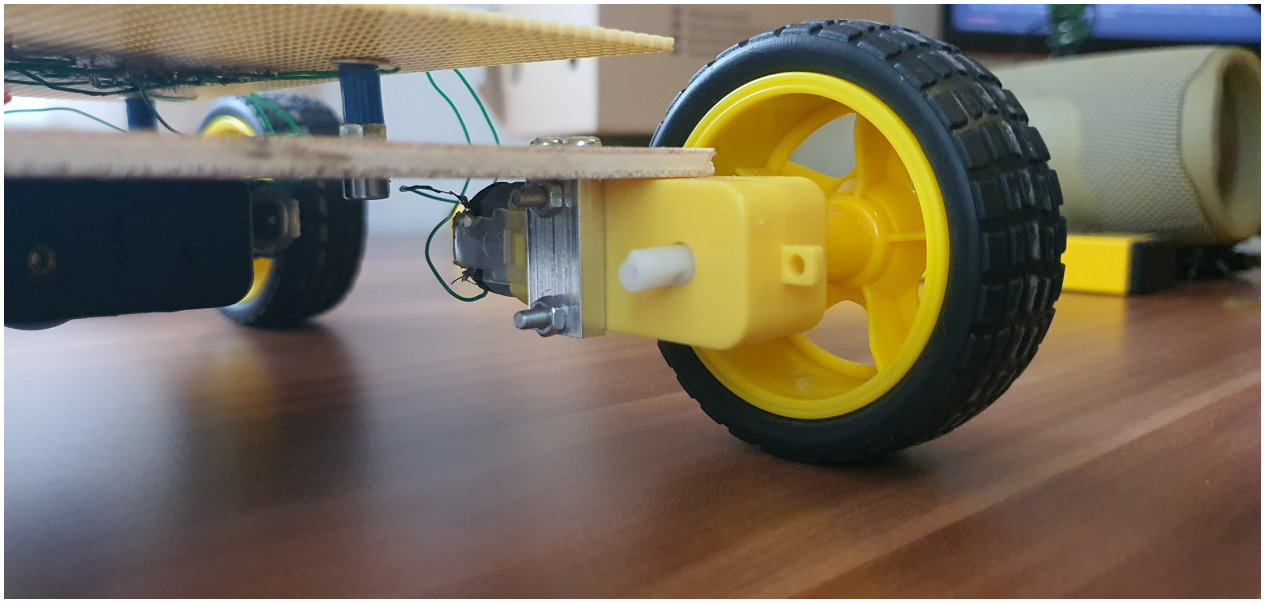
batteries:



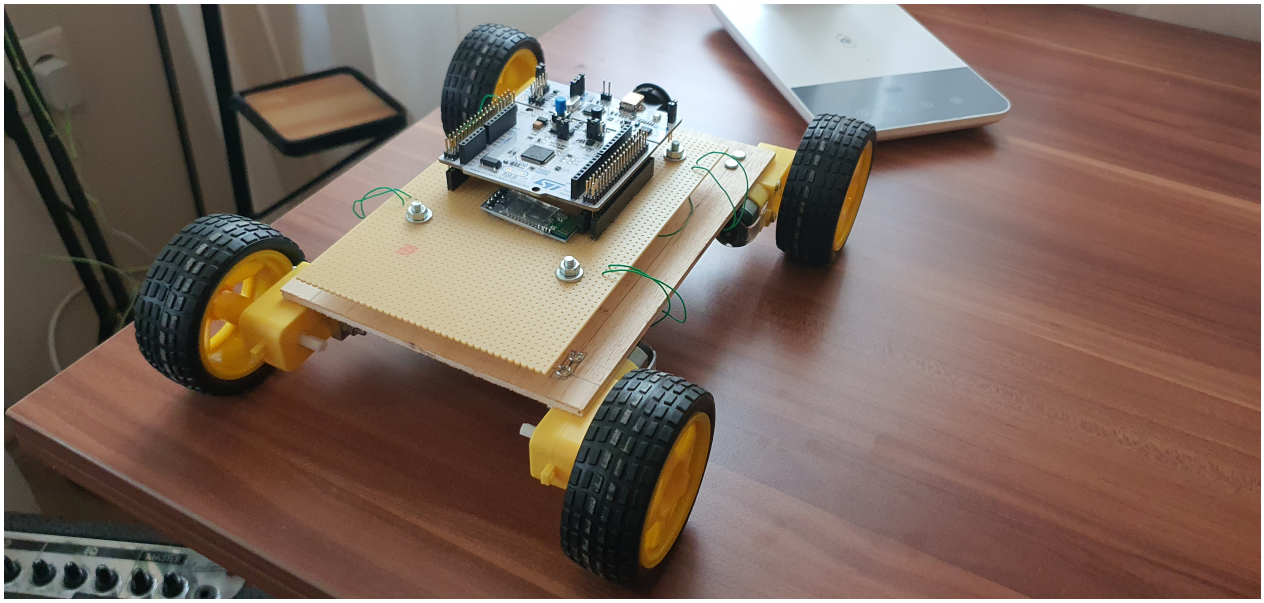
power switch:



motors:



the finished hardware:



Software

The main software was created, using STMicroelectronics `CubeIDE`. Initial code generation was done with `Cube-MX`. HAL drivers are used in the whole project.

Used peripherals:

- USART2: USART over USB debug interface
- USART3: Bluetooth communication interface
- TIMER2: PWM generation for the left side wheels
- TIMER3: PWM generation for the right side wheels
- GPIO: Direction control for all motors

`USART2` and `USART3` modules are used in interrupt mode, so the processor will be available if any other task is to be implemented in the future.

Control system

The control system recognize two sides of the vehicle. (not individual motors) Functions are implemented to set the speed, and direction of the sides.
Upon receiving a character from the serial terminal (bluetooth), the controller decides what to do in a switch-case block. The main status stored in memory is only speed, no direction information is stored.

Enums, are created for readability and easier understanding:

```
typedef enum DirectionTypeDef{
    FORWARD,
    BACKWARD
}DirectionTypeDef;

enum speed{
    slow = 30,
    normal = 60,
    fast = 90
}speedMode = slow;
```

The `DirectionTypeDef` is a simplification in the following functions, to handle directions easily.

the `speedMode`, is a global variable containing information about the speed of the vehicle, and defining legal speed values. Speeding is implemented, as setting the duty cycle for each timer, the values are represent percentages.

Functions available:

```
static void Stop(void);

inline static void Set_Speed_Right(uint8_t);        // 0 <= param < 100
inline static void Set_Speed_Left(uint8_t);            // 0 <= param < 100

inline static void Set_Speed(uint8_t);

static void Set_Dir_Right(DirectionTypeDef);
static void Set_Dir_Left(DirectionTypeDef);
```

The functions do exactly what their name says:

- `Stop()` is stopping every mortor, and set 0 duty cycle.
- `Set_Speed()` is a generalization of right and left speed setter functions

In theory, the speed functions, accept any integer between 0 and 100, but controlling scheme only works with the given values.