Università Degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Embedded Systems Course

Ro.Ve.R Robotic Vehicle off Road

Author:
Di Prisco Giovanni

Supervisor:
Vento Mario
Carletti Vincenzo

Contents

1 Introduction

The purpose of this document is to provide a description of planning and construction of an 4WD Rover as part of Embedded systems course, by giving on overview of both software and hardware architecture.

1.1 Goals and Requirements

This project deals with building an FWD rover that can work in two different modes:

- Autonomous: the rover has to move autonomously following a line drawn on the roadway.
- Manual: the rover has to be controlled by an external device using a serial communication protocol.

Furthermore, the developed firmware has to:

- manage the wheels speed rotation requested by the user so that the rover can move at a steady speed, regardless of the road gradients.
- provide a serial communication protocol that allows to choose the driving mode, to give driving commands and finally to manage sensors and actuators linked to the rover.

1.2 Project Description

The rover has four 12V motors, each of which equipped with a 2 channels Encoder. A serial asynchronous protocol is used to drive each motor separately and to provide a feedback action to control the desired speed.
The rover has also an accelerometer, a gyroscope, three reflectance sensors and an RGB LEDs matrix. Lastly it's powered by two 11.1V LiPo batteries and it has a metal shell and rubber wheels.

1.3 Design choices

The most important design choices cover motor drivers management, user communication device and the choice of the position of lane sensors.
The motor drivers provide PWM, simplified serial and packetized serial operating modes. PWM and simplified serial modes suffer from electrical disorders that could make the transmission unreliable; moreover, PWM mode needs a separate channel for each driver and so it couldn't be used because of physical limitations of the MCU

pinout. So the approach chosen is the packetized serial mode, because it provides a control over the transmission through a checksum with a default mask that makes the communication noiseless. In addition, the packetized mode provides a motor speed resolution set bigger than the simplified one's.
A Bluetooth Low Energy (BLE) has been used for communication between the rover and the user because it provides a low energy consumption(15mA), more speed (1Mbps) and a very good operating distance(50m).
The lane sensors have been placed under the rover's shield between the front wheels because this position gives the rover improved performances while following roadway line.

2 Hardware architecture

In this chapter the hardware architecture will be shown: the main hardware components will be introduced and the sensors and actuators used in the project will be described.

2.1 STM32-F401RE Board

The board used, STM32-F401RE, is shown below, along with its pinout diagram.

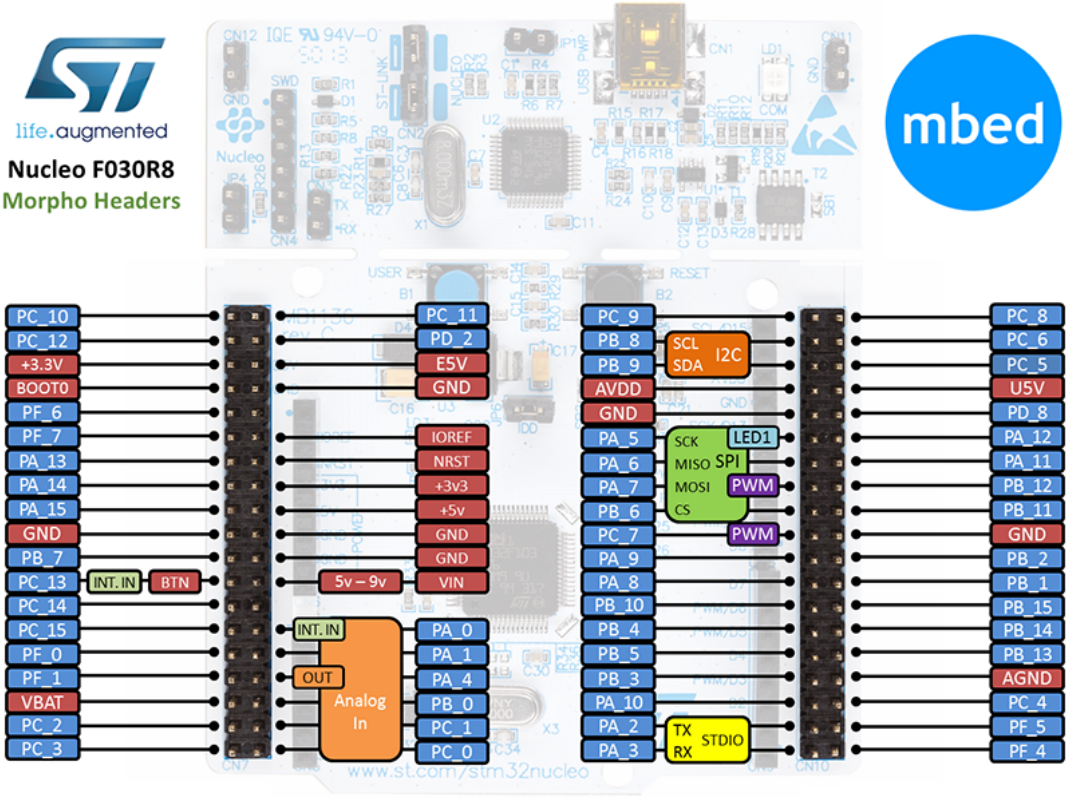

Figure 1: STM32 F401RE

2.2 Sensors

2.2.1 Pololu QTR-1A Reflectance Sensors

The Pololu QTR-1A reflectance sensor carries a single infrared LED and phototransistor pair. The phototransistor is connected to a pull-up resistor to form a voltage divider that produces an analog voltage output between 0 V and VIN (which is typically 5 V) as a function of the reflected IR. Lower output voltage is an indication of greater reflection.

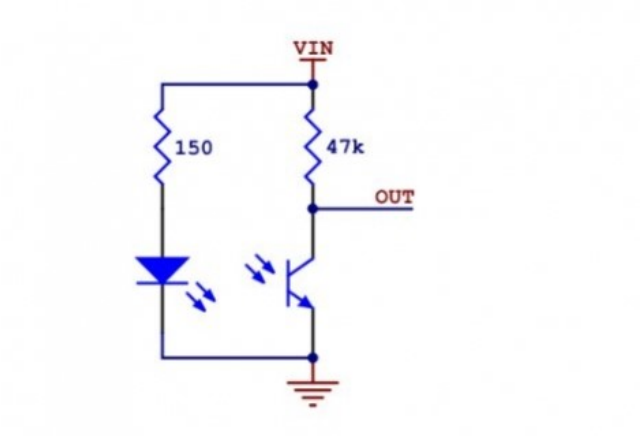

Figure 2: QTR-1A Reflectance Sensor Schematic

The LED current-limiting resistor is set to deliver approximately 17 mA to the LED when VIN is 5 V. The current requirement can be met by some microcontroller I/O lines, allowing the sensor to be powered up and down through an I/O line to conserve power.

This sensor was designed to be used with the board parallel to the surface being sensed. The sensing distance used is 0.125” (3 mm).

Figure 3: QTR-1A Reflectance Sensor

An analog-to-digital converter (ADC) has been used to measure the voltage.

2.2.2 X-NUCLEO-IKSO1A2 MEMS

The X-NUCLEO-IKS01A2 is a motion MEMS and environmental sensor expansion board for the STM32 Nucleo. It is equipped with Arduino UNO R3 connector layout, and is designed around the LSM6DSL 3D accelerometer and 3D gyroscope, the LSM303AGR 3D accelerometer and 3D magnetometer, the HTS221 humidity and temperature sensor and the LPS22HB pressure sensor. The X-NUCLEO-IKS01A2 interfaces with the STM32 microcontroller via the I2C pin, and it is possible to change the default I2C port. Among the various sensors available on the Expansion board, the rover uses LSM6DSL MEMS 3D accelerometer and 3D gyroscope.

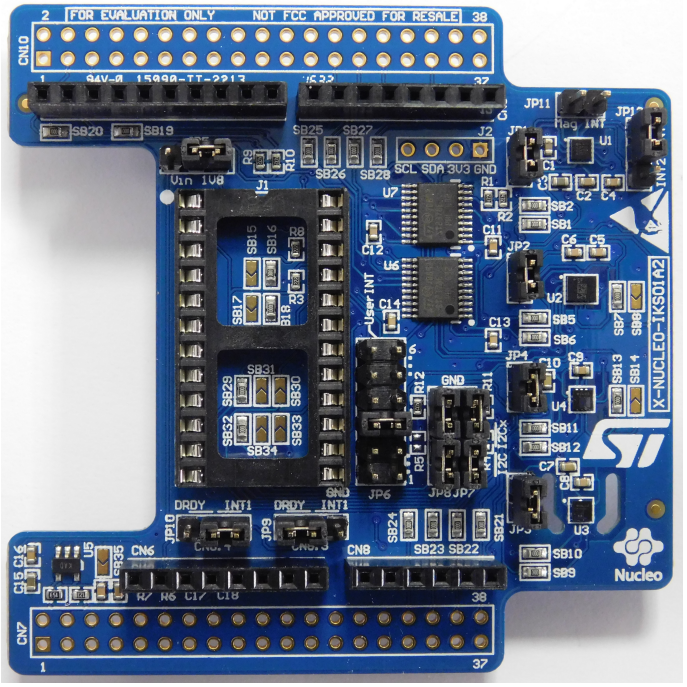

Figure 4: X-NUCLEO-IKSO1A2

2.3 Actuators

2.3.1 DimensionEngineering Driver Sabertooth 2x5

Sabertooth 2X5 is the motor driver of choice for small differential-drive robots. It is ideal for smaller robots- up to 3lbs in combat or 25 lbs for general purpose use. The Sabertooth can supply two DC brushed motors with up to 5A each continuously. Peak currents of 10A are achievable for short periods. It incorporates soft current limiting and thermal protection. It also features a lithium cutoff mode allowing

Sabertooth to operate safely with lithium ion and lithium polymer battery packs - the highest energy density batteries available.

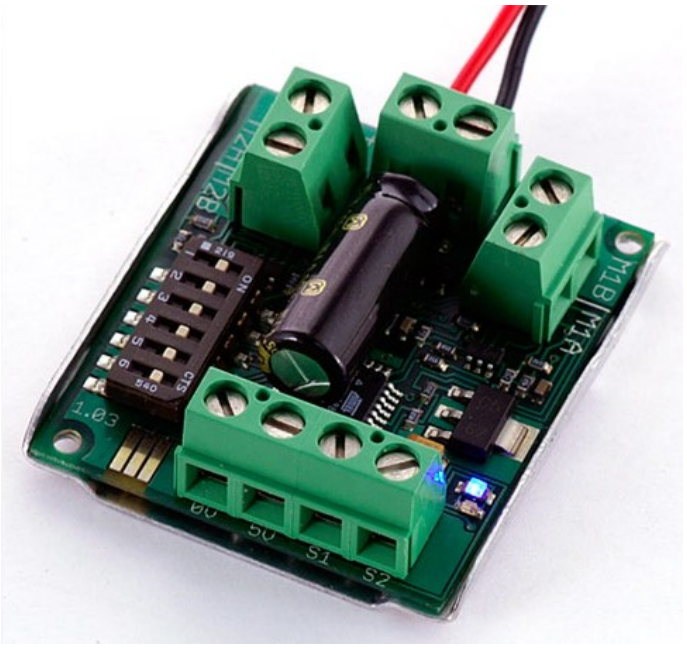

Figure 5: DimensionEngineering Driver Sabertooth 2x5

2.3.2 Pololu 2822 Metal Gearmotor with 64 CPR Encoder

Pololu 2822 Metal Gearmotor is a powerful 12V brushed DC motor with a 18.75:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 64 counts per revolution of the motor shaft, which corresponds to 1200 counts per revolution of the gearbox's output shaft. These units have a 16 mm-long, 6 mm-diameter D-shaped output shaft.

A two-channel Hall effect encoder is used to sense the rotation of a magnetic disk on a rear protrusion of the motor shaft. The quadrature encoder provides a resolution of 64 counts per revolution of the motor shaft when counting both edges of both channels.

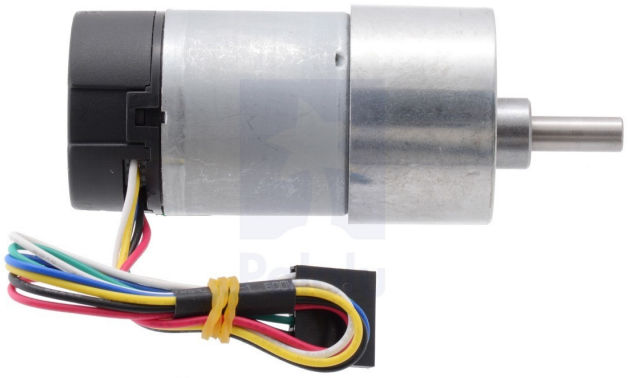

Figure 6: Pololu 2822 Metal Gearmotor with 64 CPR Encoder

The Hall sensor requires an input voltage, Vcc, between 3.5 and 20 V and draws a maximum of 10 mA. The A and B outputs are square waves from 0 V to Vcc approximately 90 degrees out of phase. The frequency of the transitions tells you the speed of the motor, and the order of the transitions tells you the direction. The following oscilloscope capture shows the A and B (yellow and white) encoder outputs using a motor voltage of 12 V and a Hall sensor Vcc of 5 V:

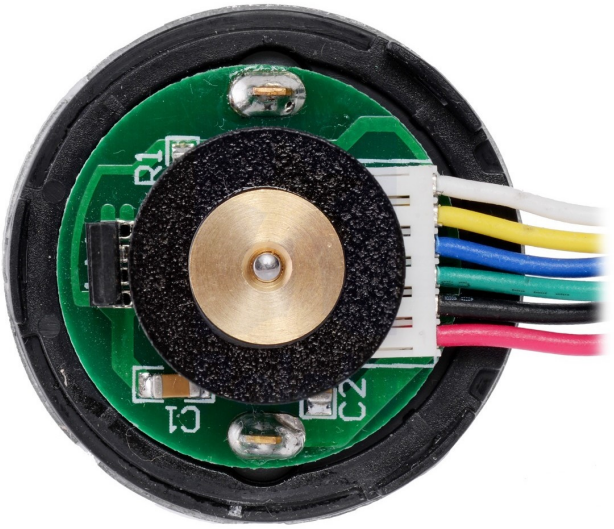

Figure 7: Pololu 64 CPR Encoder

By counting both the rising and falling edges of both the A and B outputs, it is possible to get 64 counts per revolution of the motor shaft. Using just a single edge of one channel results in 16 counts per revolution of the motor shaft, so the

frequency of the A output in the above oscilloscope capture is 16 times the motor rotation frequency.

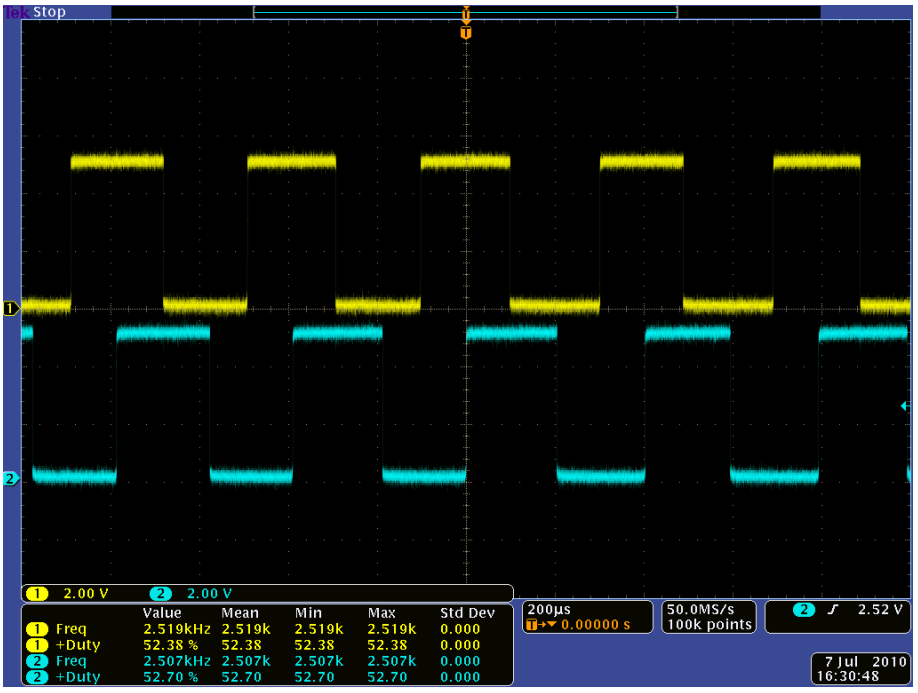

Figure 8: Encoder A and B outputs for 37D mm metal gearmotor with 64 CPR encoder

At 500 RPM this gearmotor consumes 300 mA free-run, 84 oz-in (6 kg-cm) and 5 A stall.

2.3.3 Adafruit NeoPixel featherwing RGB matrix

The Adafruit NeoPixel featherwing ia a 32 configurable eye-blistering RGB LEDs. Arranged in a 4x8 matrix, each pixel is individually addressable. Only one pin is required to control all the LEDs. On the bottom we have jumpers for the DIN line to any of the I/O pins on a Feather.

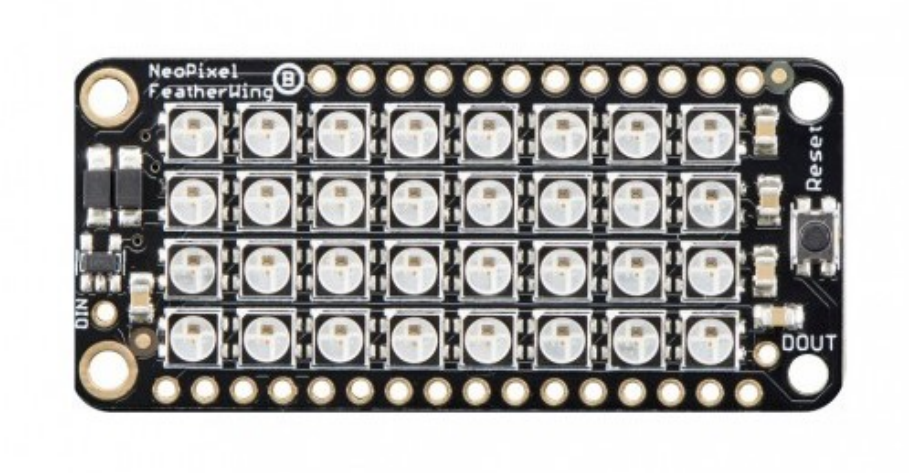

Figure 9: Adafruit NeoPixel featherwing RGB matrix

This power arrangement is able to handle 1 Amp of constant current draw and maybe 2A peak, so not a good way to make a flashlight. It's better for colorful effects.

2.4 Keyestudio Bluetooth-4.0 v2

keyestudio HM-10 Bluetooth-4.0 V2 adopts TI CC2541 chip and configuration space of 256Kb. It supports AT command.

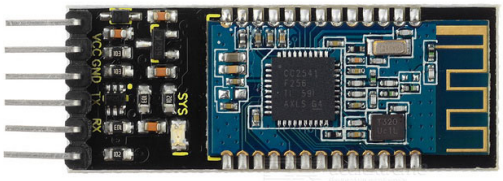

Figure 10: Keyestudio Bluetooth-4.0 v2

2.5 Electrical schematics

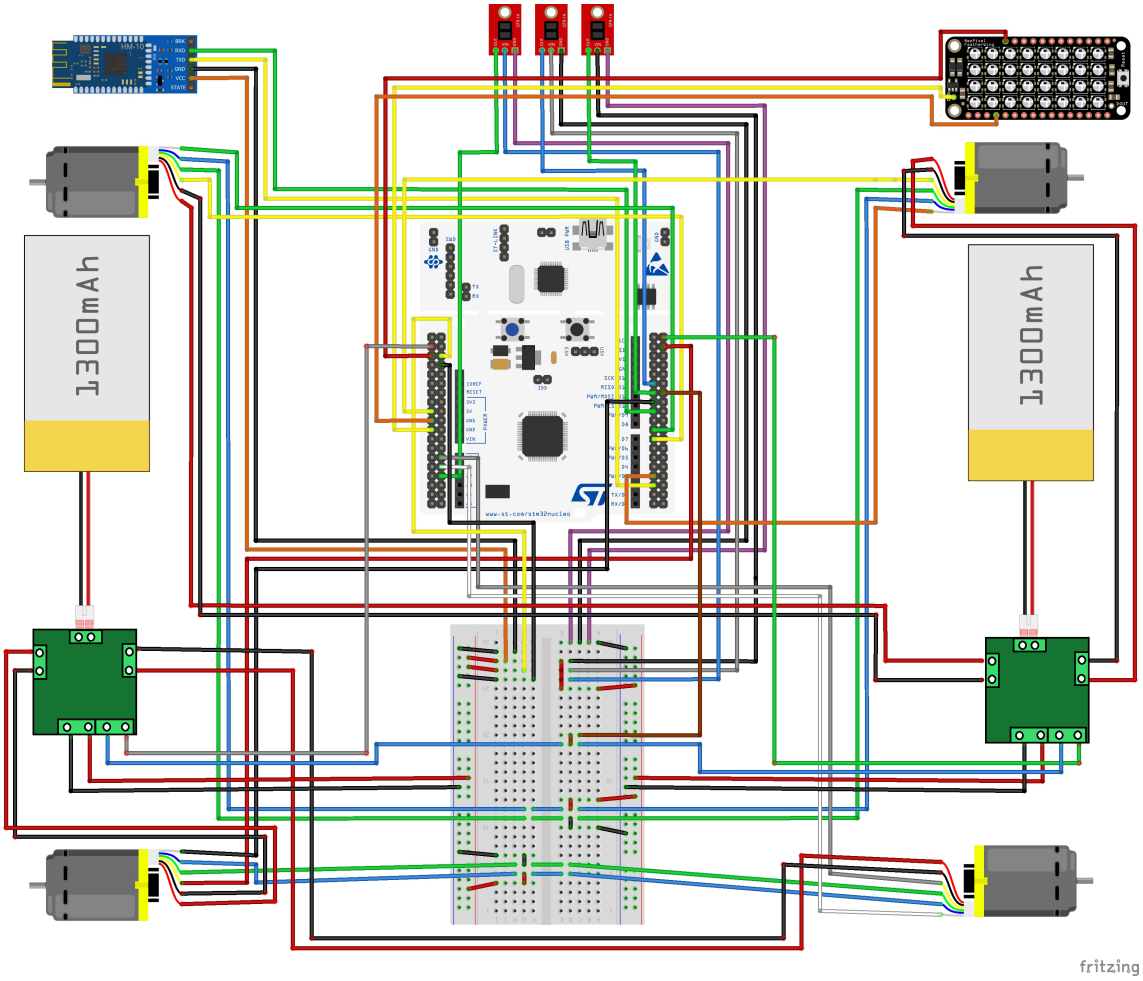

Figure 11: Electrical schematic (without X-NUCLEO-IKSO1A2 board)

List of connections and relative color of the wires:

STM32 F401RE PIN	Wire color	Device	Device PIN
PB7	Yellow	RGB LED Matrix	IN
PC12	Grey	Driver1	S2
PC8	Green	Driver2	S2
PA8	Yellow	Motor A	Encoder 1
PA9	Green	Motor A	Encoder 2
PA15	Yellow	Motor B	Encoder 1
PB3	Orange	Motor B	Encoder 2
PC6	Red	Motor C	Encoder 1
PA7	Black	Motor C	Encoder 2
PA0	Grey	Motor D	Encoder 1
PA1	White	Motor D	Encoder 2
PA11	Brown	Driver1/2	S1
PA10	Yellow	BLE	TX
PB6	Green	BLE	RX
PA6	Green	Lane sensor right	OUT
PA5	blue	Lane sensor center	OUT
PA4	purple	Lane sensor left	OUT

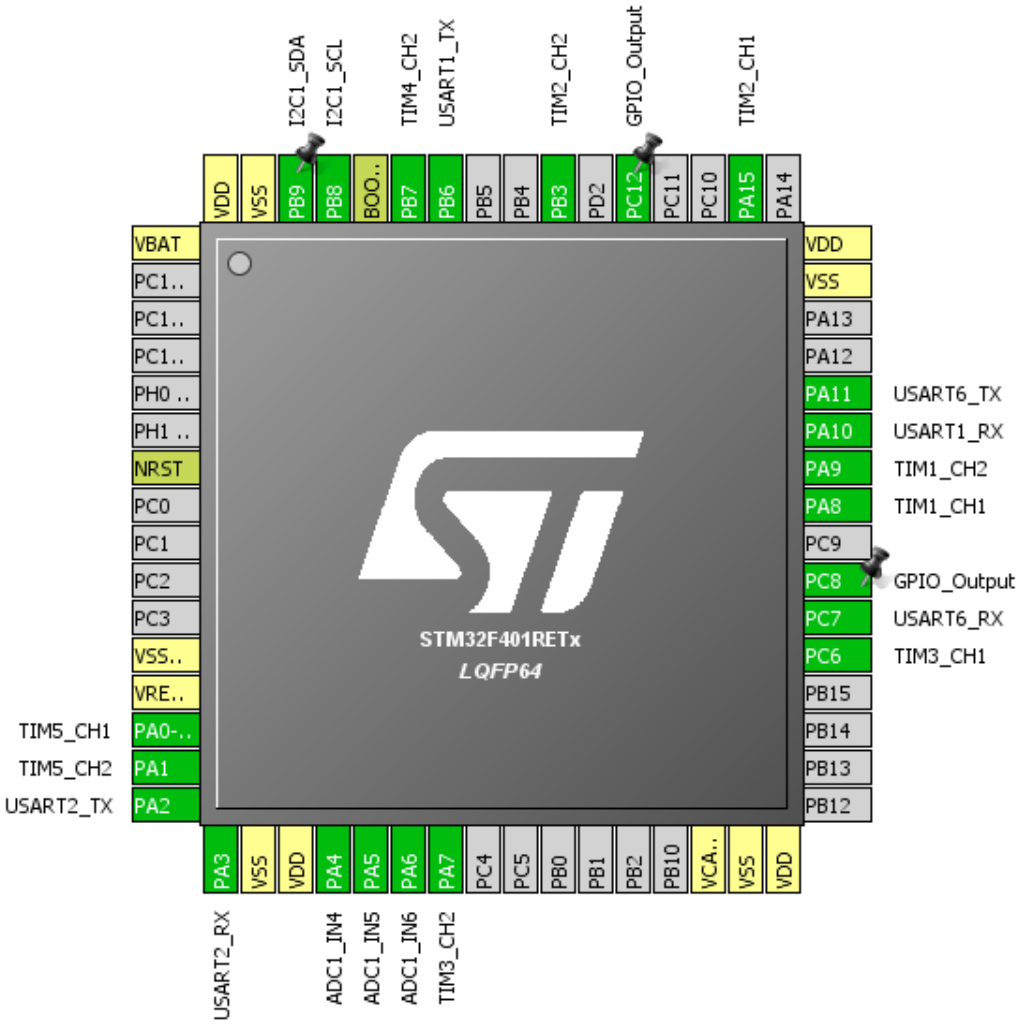

Figure 12: MCU schematic

3 Software architecture

In this chapter the software architecture will be shown: the main software functionalities will be introduced with a particular focus on the business logic of the project. Because of software complexity, only the main parts of the architecture will be described.

3.1 Business logic

3.1.1 User communication interface

A bidirectional communication with the user is made through a Bluetooth protocol: it's possible to send commands to the rover to set both the driving mode and the light intensity of the Leds matrix. It's possible to retrieve informations about linear and angular speed, acceleration and informations from lane sensors. The communication protocol provides also an emergency mode to shut the motors down instantaneously that could be disabled sending one of the generic driving commands. The protocol is composed by 4 characters, more detailed description is shown below:

- Fxxx : forward with xxx speed
- bxxx : back with xxx speed
- Lxxx : forward turn left with xxx speed
- Rxxx : forward turn right with xxx speed
- lxxx : back turn left with xxx speed
- rxxx : back turn right with xxx speed
- Axxx : autonomous drive forward with xxx speed
- Sxxx : emergency stop all
- H00x : light intensity in x mode
- Plan : show lane sensor informations
- Pagy : show accelerometer and gyroscope informations
- Pspe : show speed informations

3.1.2 Motor Control

Motor control consists of detecting speed through 2 channels timers in encoder mode. The speed is averaged and it is sampled every 88ms. Wheels direction is detected through timers position.
Error feedback is carried out from the difference between the desired speed and the speed produced by the encoders, calculated in RPM. A PID(proportional-integrative-derivative) controller has been implemented using the Ziegler-Nichols method. The Ziegler–Nichols tuning method is a heuristic method of tuning a PID controller, in order to retrieve the PID parameters and it is performed as follows:

- Setting the I (integral) and D (derivative) gains to zero;
- The "P" (proportional) gain, Kp, is then increased (from zero) until it reaches the ultimate gain Ku, at which the output of the control loop has stable and consistent oscillations;
- Ku has been calculated as 3/5 of Kp;
- Ku and the oscillation period(TU) are used to set the P, I, in particular KI = 2*Kp/TU and Kd = TU*KU/8;

This tuning rule is meant to give PID loops best disturbance rejection. Moreover, through further experiments a wind-up value has been calculated in order to avoid that a large change in setpoint occurs and the integral terms accumulates a significant error during the rise, overshooting and continuing to increase as this accumulated error is unwound.

And finally, because the motor control mode provides for a set of 128 speed values, the feedback speed is converted according to this resolution set.

```
int16_t PID(int16_t speed, int16_t*last_error, int16_t*integral,
        int16_t*derivative, int16_t desired_speed) {
    int16_t error = desired_speed - speed;

    if (((*integral) >= WINDUP) && (error < 0)) {
        *integral += error;
    } else if (((*integral) <= (-WINDUP)) && (error > 0)) {
        *integral += error;
    } else if (((*integral) >= (-WINDUP)) && ((*integral) <= WINDUP)) {
        *integral += error;
    }

    *derivative = error - *last_error;

    int16_t trigger = desired_speed + KP * error;
    if (abs(desired_speed) <= 60) {
        trigger += KI_L * (*integral) + KD_L * (*derivative);
    } else {
        trigger += KI_H * (*integral) + KD_H * (*derivative);
    }
    *last_error = error;
    return calculate_code_speed(trigger);

}
```

Figure 13: PID method

3.1.3 Self-driving

A self-driving algorithm has been implemented by processing the data coming from the three lane sensors. It's based on the idea that the rover either has to go ahead if the central lane sensor is detected or it has to make small corrections through a differential steering if two sensors are detected or it has to make large corrections through a stronger differential steering system if only a side sensor is detected. So far as the sensors don't detect the lane anymore, the algorithm tries to correct rover's behavior according to the previous detection, reporting a warning signal: a timeout event will occur if the sensors don't detect the lane in 3 seconds.
The following table describes this algorithm in more detail.

Sensor reading behavior:

Lane detector	Action	Previous state	Refresh Timer	Warning
111	Stop	Reset	N	Y
110	Weak differential steering to the left	LEFT	Y	N
011	Weak differential steering to the right	RIGHT	Y	N
001	Strong differential steering to the right	RIGHT	Y	N
100	Strong differential steering to the left	LEFT	Y	N
000			N	Y

Behavior if no sensor detects the lane:

Previous state	Action	Previous state	Refresh Timer	Warning
LEFT	Slow turn to the left		N	N
RIGHT	Slow turn to the right		N	N
TIMEOUT	Stop	Reset	N	Y

3.1.4 MEMS interface

3D linear and angular acceleration (on X,Y,Z axis) can be displayed from LSM6DSL sensors, according to user request.

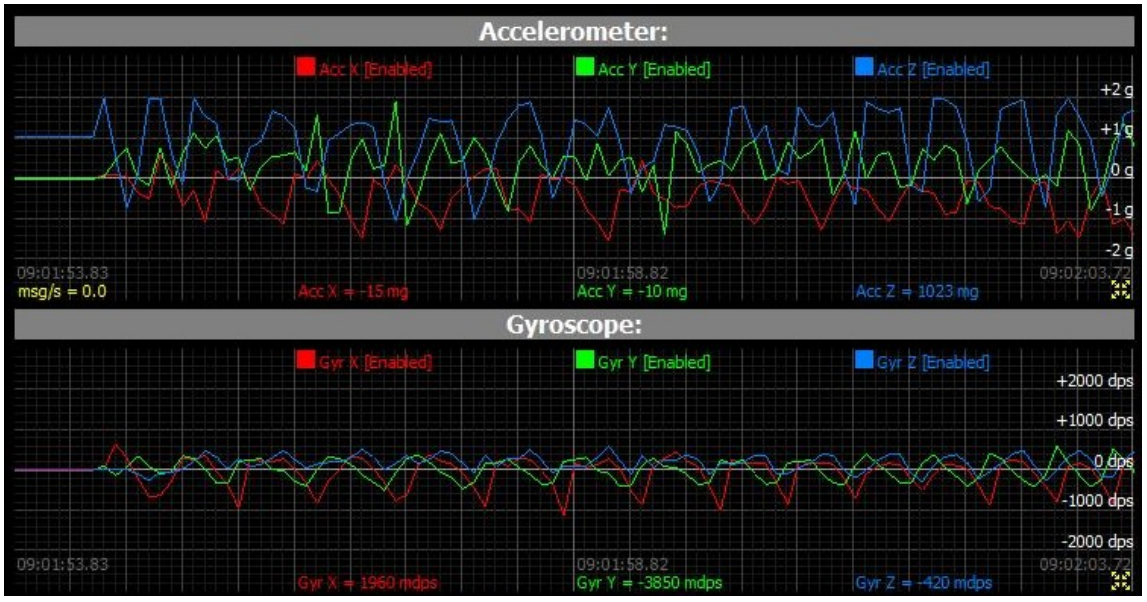

Figure 14: 3D accelerometer and 3D gyroscope values

3.1.5 NeoPixel LED matrix management

The NeoPixel FeatherWing RGB LEDs matrix provides seven operating modes:

- Break (red light)
- Headlight level 0 (turned off)
- Headlight level 1 (white light, 6 percent of the brightness)
- Headlight level 2 (white light, 50 percent of the brightness)
- Headlight level 3 (white light, 100 percent of the brightness)
- Rear light (white light with a red frame)
- Warning (orange light)

The matrix reacts every 500 ms to the rover state: if every wheel is fixed, matrix is in break state; according to the driving direction it acts like headlight or rear light; the warning state has priority over the other states. The user can manage the four headlight levels in any driving mode.

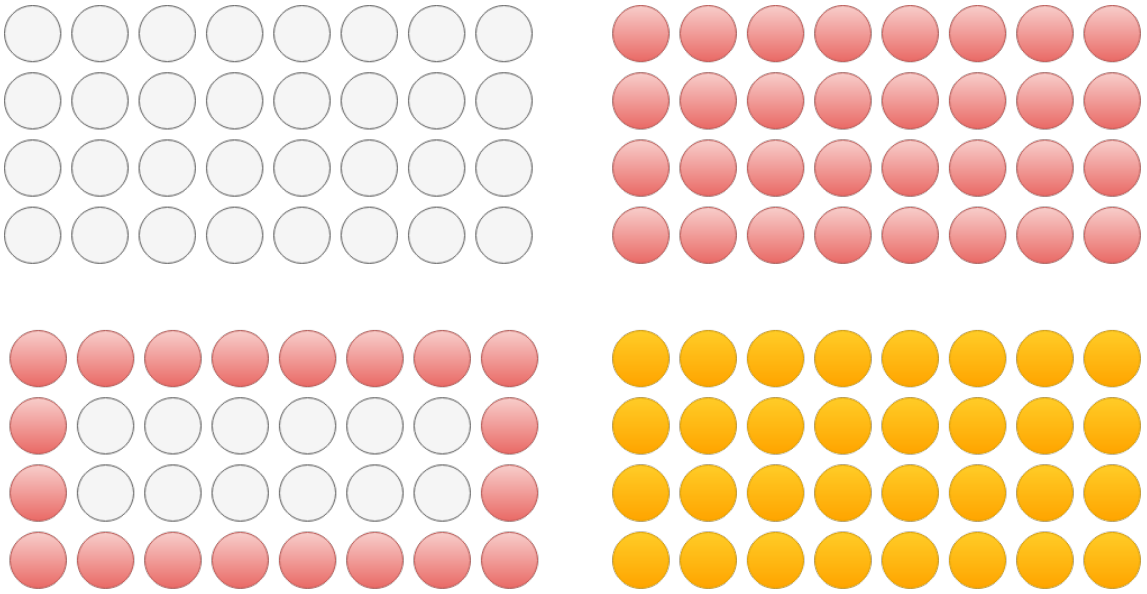

Figure 15: NeoPixel RGB LED matrix states

4 Communication interfaces

This chapter will introduce which communication interfaces have been implemented and so how the rover communicates with the user.

4.1 BLE USART Interface

As said previously, a user communication interface has been developed. In particular a BLE USART Interface has been developed, using the functionalities provided by the GPIO pins of the STM32-F401RE (USART1). The Baud rate related to the UART is 9600 bit/s. The data is acquired by the MCU through interrupt mechanism.

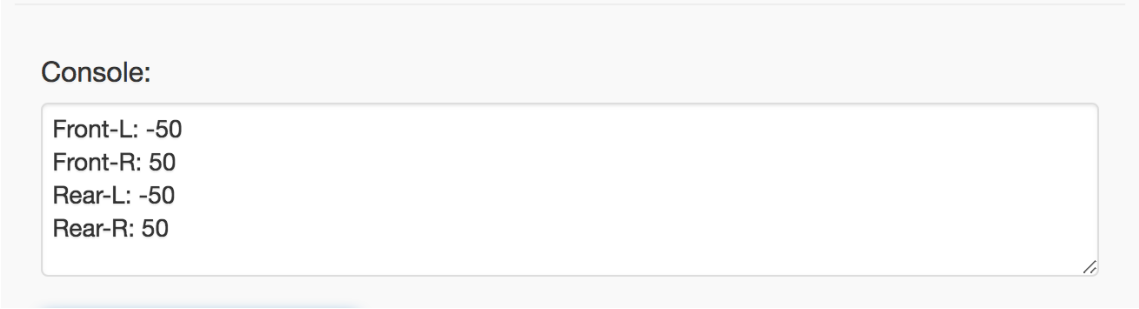

Figure 16: Example of message received via BLE

4.2 Debug USART through USB

Another USART communication interface(having baud rate equal to 9600 bit/s) is provided for debug purpose. It is an unidirectional interface aimed to send the values coming from sensors (such as speed in RPM, lane sensors threshold etc.), feedback about speed and the speed given by the user input.

5 Supporting software

5.1 STM32CubeMX

STM32CubeMX is part of STMicroelectronics STMCube™ original initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio. STM32Cube includes STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
It also embeds comprehensive STM32Cube MCU Packages, delivered per STM32 microcontroller Series (such as STM32CubeF4 for STM32F4 Series). These packages include the STM32Cube HAL (an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 portfolio), the STM32Cube LL (low-layer APIs, a fast, light-weight, expert-oriented layer), plus a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics. All the embedded software utilities are delivered with a full set of examples.
STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and the generation of the corresponding initialization C code through a step-by-step process.

5.2 System Workbench for STM32

The System Workbench toolchain, called SW4STM32, is a free multi-OS software development environment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards.

5.3 Remote Controller App

Remote controller is an application which allows to manage the rover remotely. We can see it as a web app enjoyable by any device capable to run a browser, like smartphones, tablets and PCs. Written in a responsive way, the controller will adapt based on the device on which it runs providing so an user-friendly interface the user can interact with.

On the server side, there is a Node js server which manage the bluetooth connection with the rover and communicate to it based on the messages the web application requests.

So, using the Remote Controller App the user can interact with the rover, sending commands like "go straight with speed x" or "turn the light on" but also "activate

the autopilot". A feedback is provided for each command sent. The user can also ask for information about the sensors: Speed, Lane Detector, Acc/Gyro.

5.3.1 Technology Stack

Following the technology used for this application:

Frontend: HTML, CSS, Bootstrap, Javascript

API Layer: Node js, Javascript

Backend: Node js, noble

5.3.2 SW Architecture

In this paragraph the sw architecture is described.

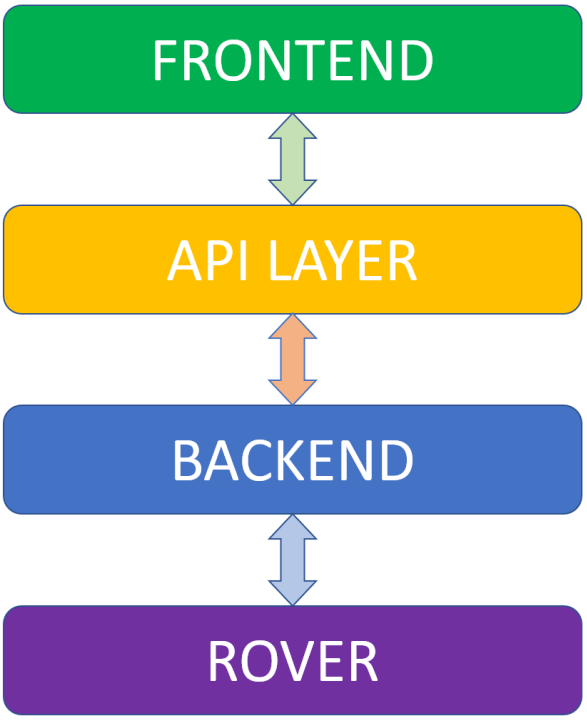

Figure 17: SW Architecture

Frontend This is what the user see. It is basically an HTML page written with bootstrap and javascript.

Figure 18: Web App running on smartphone

Api layer This component allows the frontend to send and receive message to/from the backend. 2 apis are provided:

- **GET** /status/infoConnection
- **GET** /controller/:message

Backend This component manage the connection with the rover and forward to it any message the api layer receive and then send the response/feedback back to the frontend.

5.3.3 Features

Within the application, user can:

- turn the light on/off and set its intensity
- set the auto mode
- turn the emergency mode on
- set the direction of the rover
- set the speed
- get info about the sensors

Light There is a slider on which the user can select a number of the following [0,1,2,3]. By selecting 0 the light will be turned off. On the other hand, selecting 3 will turn the light at the full intensity

Auto mode Tapping on the **A** button, the rover will run in Auto mode.

Emergency mode Tapping on the **red** button, the emergency command will be sent to the rover.

Speed The speed is selected using a slider on which is allowed to select a number in the range 0, 500. The speed is set when any of the *Direction* button is pressed.

Direction 6 buttons are provided to the user.
There are 2 categories: up and down buttons. Using them, the user can set the direction the rover will follow, and using the speed properly is then possible to lead the rover to any direction with any rotation angle without any speed limit.

Sensors The rover is equipped with 3 sensors: Speed, Lane Detector, Acc/Gyro. For each of them a button is associated and by clicking it the user will receive information about the related sensor. The output is visible inside the **Console**.

6 Images

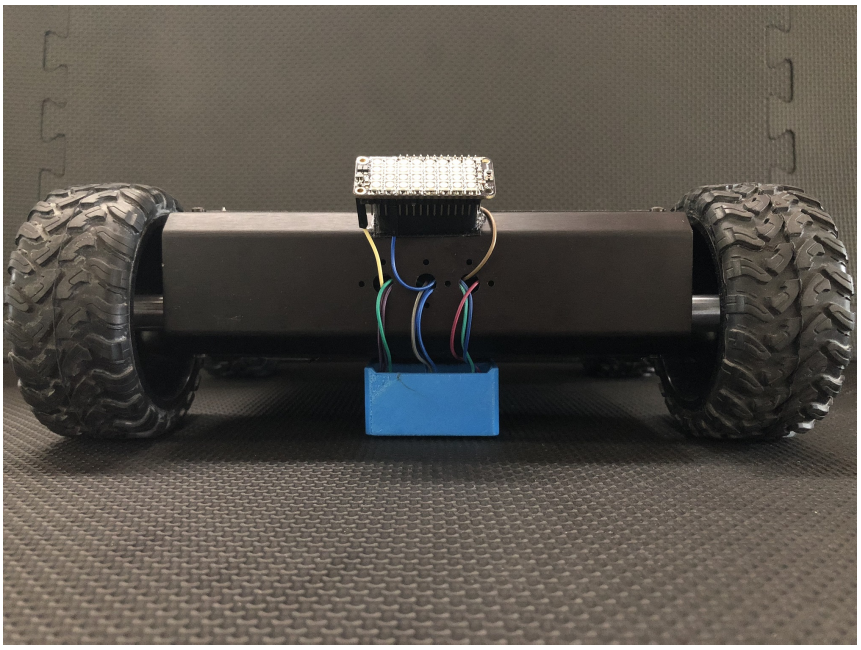

Figure 19: Rover: front

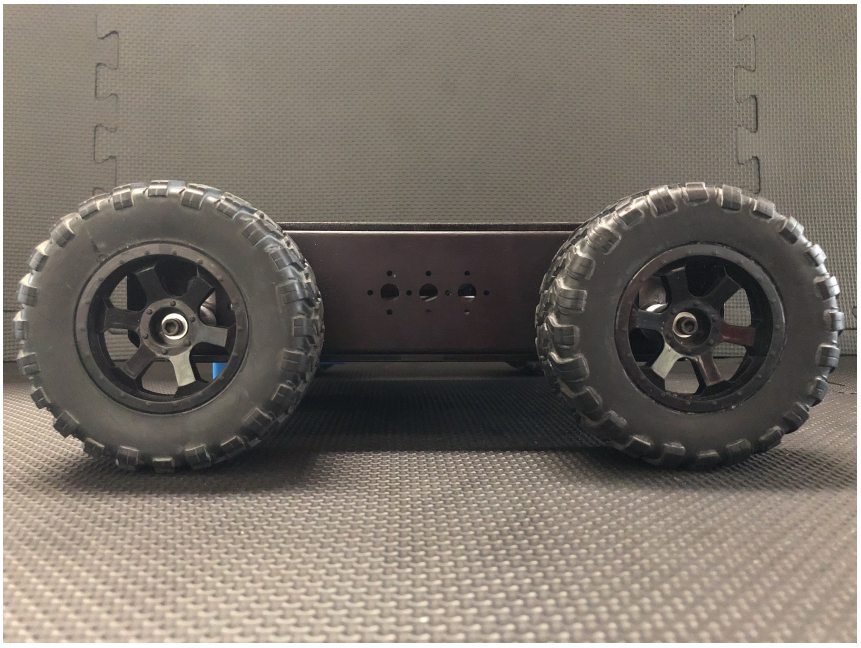

Figure 20: Rover: side

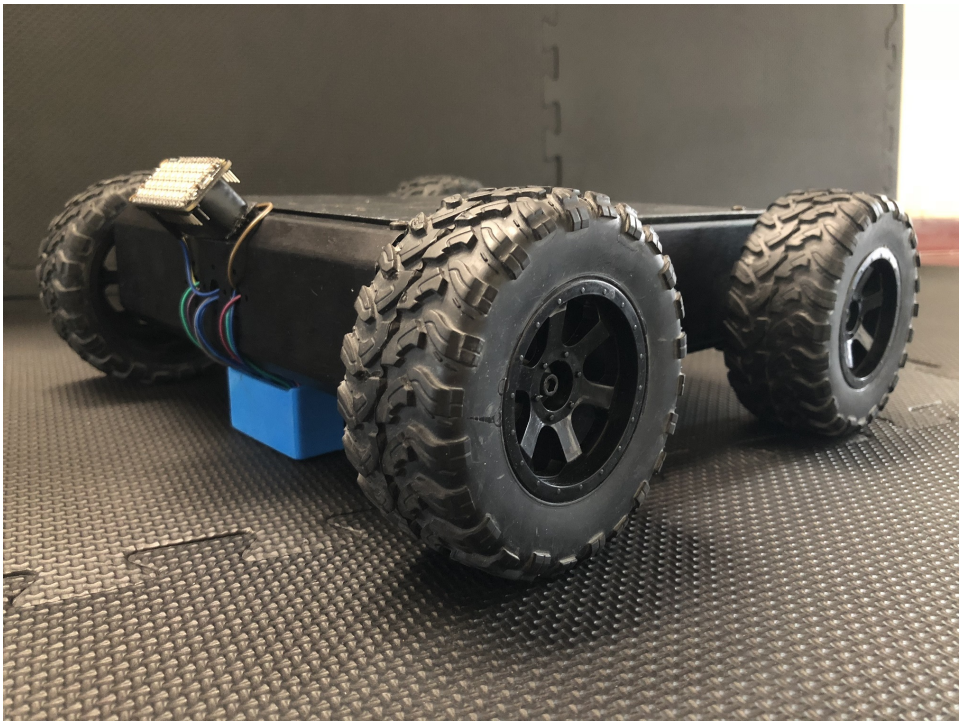

Figure 21: Rover: detail of the front

Figure 22: Rover: detail of the rear

Figure 23: Rover: stopped in downhill

Figure 24: Rover