UNIVERSITA' DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
ED ELETTRICA E MATEMATICA APPLICATA

PROJECT DOCUMENTATION

Title:

UOLL – I

Authors:
Caruso Oscar
Longobardi Francesco
Montillo Andrea
Sicignano Andrea

Index

Introduction

Aim of the document

The document has the purpose of validating the planning work and the development of a robot whose name is UOLL-I, chosen as final project of Embedded System course. This is the official documentation and it has the aim of describing the functional specifications of the system as well as the hardware components and software architecture.

Aim of the project

Basically, the robot has a playful purpose, the user can play with it joining different functionalities. It has the function of collecting and preserving objects inside its own body. A Bluetooth Application is used to drive the robot and carry out the collection of an object. Moreover, it has a load cell through which the weight of the kept objects is measured and an LCD where are displayed useful data to user.

So, the "pick up" procedure is made of some different parts, first of all UOLL-I compute the distance of the object and go to the correct position for the right opening of belly. Then it starts moving the arms in a fixed way to push the object inside its body. Finally, it closes the belly and measures the weight of the object collected.

All the system works like a finite state machine where in each state there are some possible operation and some ignored ones. In particular, while moving it is not possible to do "pick up" or "dance" operation, also the opening of the belly is ignored because it could be dangerous to move with the gate open.

The "un-do" operation is based on the concept of cancel some movements that are going wrong. So, it stops the movements immediately and after a while it does the inverse operation.

Chapter 1: Functional specifications

The robot is controlled by the MCU STM32F401RE, programmed in C language. It works through a mobile app that allows bluetooth serial communication and through which it is possible to send commands. There is an H-Bridge that is connected with two brushed motors through which it manages the movement commands, while the orders given to open or close the gate and the motion of the arms come to the servo motors connected to some PWM pins. Moreover, there is a weight sensor used to measure the total load of the collected objects. It is very useful because, if the weight of objects is 2 Kg, the request of collect an object must be ignored. This choice is made because the maximum weight that the track can support is 5 kilos and the weight of the body, sensors, circuits, MCU, etc. is about 3 kilos. The ultrasonic sensor is used to evaluate the distance between the object and the robot using this value to better perform the "pick up" command. There is also an LCD on which useful data are printed and two RGB pins into the eyes with the aim of informing the user about actual state of UOLL-I.

1.1 How it works

At the beginning, the software proceeds to initialize the modules like: *Timer*, *UART*, *DMA*, *I^{2}C*, *GPIO*. Then, it initializes the motors, the LCD, the ultrasonic sensor and also the weight sensor which has to be calibrated to set the right value of the tare.

At the end of this initialization, the robot starts to listen for a Bluetooth command on the serial port. Then the external supply voltage value is evaluated to determine which are the commands that can be executed. The communication with UOLL-I has been facilitated by the use of a mobile application that allow to send orders by pressing the buttons on the controller. In particular, these commands are shown below:

- ↑ orders the robot to go ahead;
- ↓ orders the robot to go in reverse;
- → orders the robot to go right;
- ← orders the robot to go left;
- "START" orders the robot to start a sequence of operation (like dancing);
- "SELECT" orders the robot to stop what it is doing and it orders to come back to the initial conditions;
- "Δ" orders the robot to open or close the belly;
- "□" orders the robot to execute the "pick up" routine to collect an object;
- "○" corresponds to the letter "c" that orders the robot to execute the arm movement;
- "X" the robot to print on the display the distance of an object in front of UOLL-I and the weight inside its belly;

Chapter 2: Hardware Architecture

2.1 Description of the board

The board used to control the operation of the system was provided by STMicroelectrionics. It is equipped with a 32-bit RISC ARM® Cortex®-M4 MCU that works at the frequency of 16 MHz, 512 Kbytes Flash memory and 96 Kbytes SRAM, in addition to an extensive range of peripherals and I/O connections. Furthermore, there are six 16-bit timers and two 32-bit timers, a 12-bit ADC, a low consumption RTC and two buttons SET and RESET.

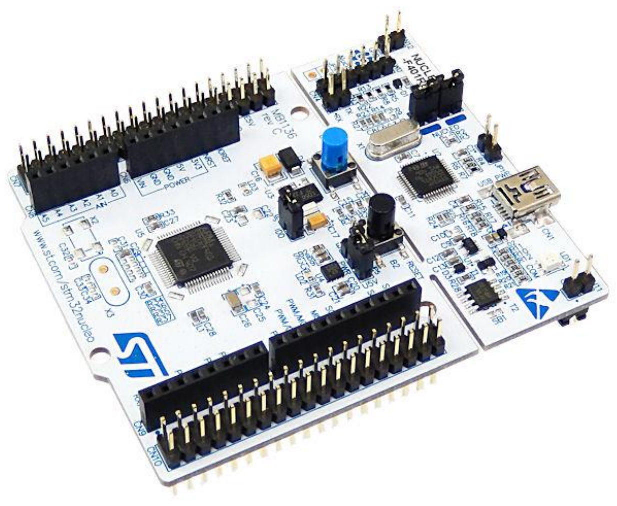


Figure 2.1: STM32F401RE board

The following table shows the configured STM32 pins:

Pin	Description	Note
PC7	Bluetooth	IN (UART6_RX)
PC6	Bluetooth	OUT (UART6_TX)
PB8	LCD	I2C1_SCL
PB7	LCD	I2C1_SDA
PC4	DT Weight	IN
PC5	SCL Weight	OUT
PC0	H-Bridge IN1	OUT
PC1	H-Bridge IN2	OUT
PC2	H-Bridge IN3	OUT
PC3	H-Bridge IN4	OUT
PA8	Servo	OUT (TIM1)
PA5	Servo	OUT (TIM2)
PA6	Servo	OUT (TIM3)
PB6	Servo	OUT (TIM4)
PA0	Servo	OUT (TIM5)
PA1	ADC	IN1

PA4	ADC	IN4
PB9	RGB LED	OUT
PA9	RGB LED	OUT
PA7	RGB LED	OUT
PA14	HC-SR04 (Echo)	IN
PA15	HC-SR04 (Trig)	OUT
PB4	First Hall Sensor	IN
PB5	Second Hall Sensor	IN

Table 2.1: STM32 configured pins

2.2 Connection Schema

The diagram of the connections made is shown below:

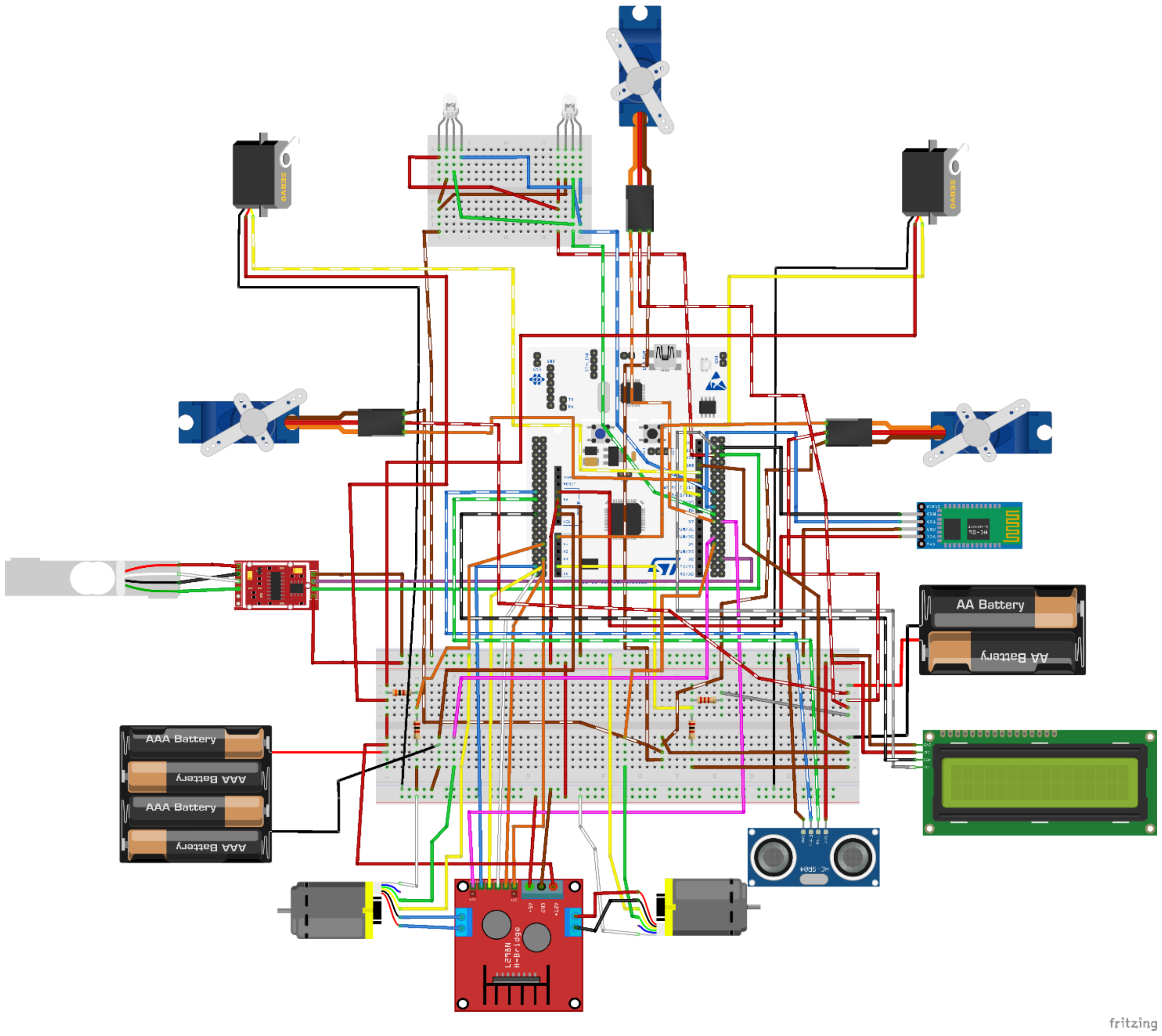


Figure 2.2: Connection Schema

2.3 Description of the components

2.3.1 Tracked car

A tracked robot smart car platform aluminum alloy chassis of the Mountain Ark has been used, with a size of 20*18.5*6 cm (l,w,h) and with weight of 870 grams. This platform allows to hold up a maximum load of 5 Kg, it has high flexibility and stability in its movements that it manages to climb a flat surface up to 45 degree without load.

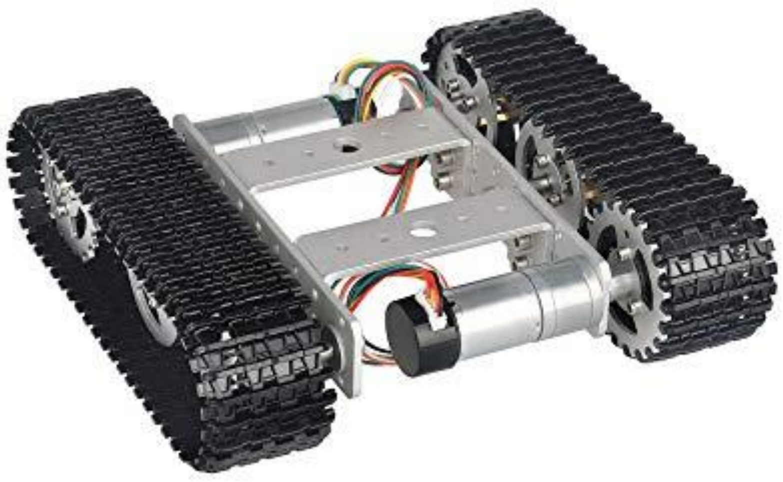


Figure 2.3: Tracked robot smart car platform aluminum alloy chassis

2.3.2 Brushed motors

Two DC brushed motors have been used, each of 25 mm powered at 9V and equipped with encoder. It is composed by an Hall sensor, one channel used to detect the rotation of the magnetic disk on the external motor shaft. The encoder has a 2-pulse resolution per revolution of the motor shaft. Moreover, through their use, the robot can evaluate how much time keep moving towards the object during "pick up" sequence. The motor speed is about 150 revolutions/minutes and it has an high torque of 9.5 Kg per cm.

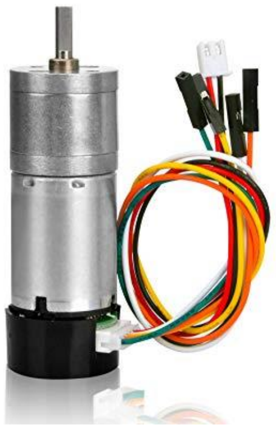


Figure 2.4: GM25-370 25mm Gear Motor

Wire Color	Function
Black	Motor V_{cc}
Red	Motor GND
White	Hall sensor V_{cc}
Yellow	Hall sensor GND
Orange	Hall sensor S1
Green	Hall sensor S2 (unused)

Table 2.2: Brushed Motor pins

2.3.3 Servo motors

The servo motors used have common features regarding the management of PWM signal with the STM32 board, as shown in the table below:

Feature	Value
Period	20 ms
Minimum Duty Cycle	0.5 ms
Maximum Duty Cycle	2.5 ms

Table 2.3: servo motors common features

2.3.3.1 MG996R Servo

The MG996R digital servo is a 25 teeth high-torque motor that can rotate approximately 180 degrees (90 in each direction). The MG996R is essentially an upgraded version of the MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. Furthermore, the gearing and motor have been upgraded to improve dead bandwidth and centering. The servo comes with 30cm wire and 3 pin 'S' type female header connector. The stalling torque results to be 9.4 - 12 Kg/cm with a required voltage between 4.8 V - 6 V. The robot has two MG996R servos driven by two timers configured in PWM mode and both placed on the elbows in order to give to the arms another level of mobility useful for the collection of objects.

Figure 2.5: MG996R

Wire Color	Function
Red	V_{cc}
Brown	GND
Orange	PWM

Table 2.4: MG996R pins

2.3.3.2 LD20-MG Servo

The LD20-MG digital servo is a 25 teeth high-torque motor that can rotate by 180 degrees in each direction. The stalling torque results to be 20 Kg/cm with a required voltage between 6 V - 7.4 V. The gearing and the motor are full metal to have better accuracy. An aluminum case enhances the heat dissipation and ensures that the servo motor can work well. Furthermore, it comes with 30cm wire and 3 pin ‘S’ type female header connector. The robot has three LD20-MG servos driven by three timers configured in PWM mode. They are used for the gate on the belly and the shoulders. In fact, these parts of the body require more power for a correct behavior.

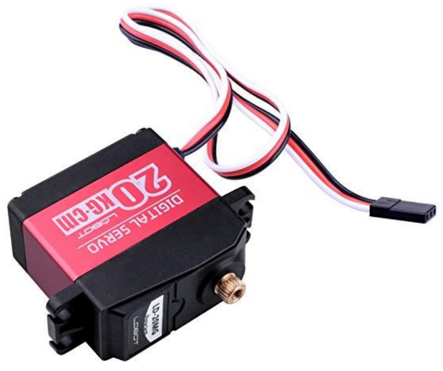


Figure 2.6: LD20-MG

Wire Color	Function
Red	V_{cc}
Black	GND
White	PWM

Table 2.5: LD20-MG pins

2.3.4 H-Bridge

The dual h-bridge L298N is a motor driver that has LED indicators, a 5 V voltage regulator and some diodes. It is ideal for robot applications and suitable for connection with microcontrollers as it requires few pins to control each motor. Inside the driver there are two integrated h-bridges that support a high voltage, up to 35 V, and high current,2 A per bridge. The h-bridges can be driven with TTL logic levels, moreover each bridge can be disabled or enabled via the corresponding enable pin to control a stepper motor or two DC motors directly from the microcontroller.

Then through the bridge, it is possible to make the robot move in all directions.

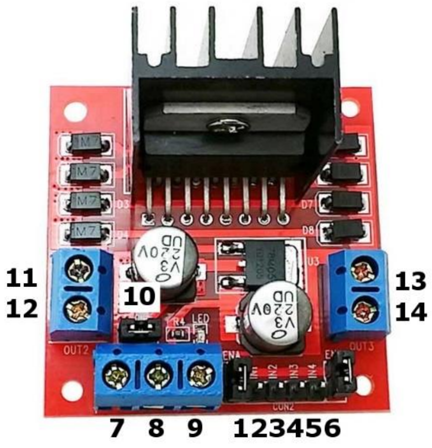


Figure 2.7: H-Bridge L298N

N°	Function
1	ENA – DC motor A enable jumper, connect to a PWM pin to control the motor speed
2	IN 1
3	IN 2
4	IN 3
5	IN 4
6	ENB – DC motor B enable jumper, connect to a PWM pin to control the motor speed
7	Motor supply voltage, maximum 35 V DC
8	GND
9	5 V power supply input, useful for powering the internal logic
10	Jumper 12 V, removed to allow the 5 V supply
11	V_{cc} of the 1st DC motor
12	GND of the 1st DC motor
13	V_{cc} of the 2nd DC motor
14	GND of the 2nd DC motor

Table 2.6: H-Bridge L298N

2.3.5 Load Cell and Weight Sensor

The load cell is composed of a metal structure and small elements called strain gauges, assembled in some points of the structure. It provides an output electrical signal which varies according to the deformation produced by the weight applied to the component. The capacity of the strain gauges is 5 Kg with a required voltage of 5 V.

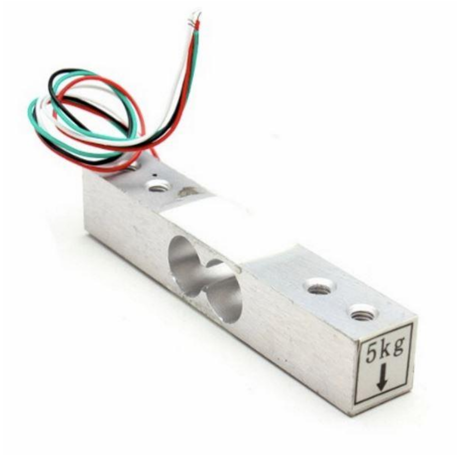


Figure 2.8: 5Kg Load Cell

Then, in order to have correct measurements, it has to be fixed on the body according to a particular structure shown in the figure below:

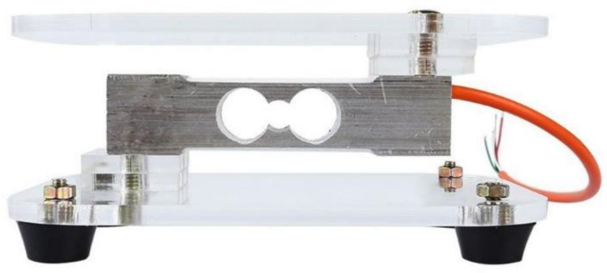


Figure 2.9: “Z” structure

Based on semiconductor’s technology, HX711 sensor is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor. The input multiplexer selects either Channel A or B differential input to the low-noise programmable gain amplifier (PGA). Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of ±20 mV or ±40 mV respectively, when a 5 V supply is connected to the analog power supply pin. Channel B has a fixed gain of 32 and the full-scale input voltage range is ±80 mV. On-chip power supply regulator eliminates the need for an external supply regulator to provide analog power for the ADC and the sensor. Clock input is flexible, it can be from an external clock source, a crystal or the on-chip oscillator that does not require any external component. There is no programming needed for the internal registers, in fact all controls to the HX711 are through the pins. The output 24 bits of data is in 2’s complement format. When input differential signal goes out of the 24-bit range, the output data will be saturated at a minimum of 0x800000 or at a maximum 0x7FFFFF, until the input signal comes back to the input range. Pin SCK input is used to power up and down the HX711, when the SCK input is low the chip is in normal working mode. When SCK pin changes from low to high and stays at high for longer than 60μs, HX711 enters power down mode. When SCK returns to low, chip will reset and enter normal operation mode. After a reset or power-down event, input selection is default to Channel A with a gain of 128. Pin SCK and DOUT are used for data retrieval, input selection, gain selection and power controls. When output data is not ready for retrieval, digital output pin DOUT is high. Serial clock input SCK should be low. When DOUT goes to low, it indicates data is ready for retrieval. By applying 25~27 positive clock pulses at SCK pin, data is shifted out from the DOUT output pin. Each SCK pulse shifts out one bit, starting with the MSB bit first, until all 24 bits are shifted out. The 25th pulse at SCK input will pull DOUT pin back to high. Input and gain selection are controlled by the number of the input SCK pulses. The pulses should not be less than 25 or more than 27 within one conversion period, to avoid causing serial communication error.

SCK Pulses	Input Channel	Gain
25	A	128 (used)
26	B	32
27	A	64

Table 2.7: Input Channel and Gain Selection

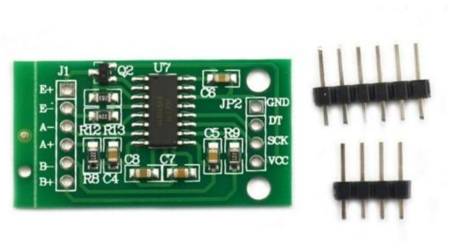


Figure 2.10: HX711 module

Wire Color	Connection
Red	E+
Black	E-
White	A-
Green	A+

Table 2.8: Connection between Load Cell and HX711

Pin	Function
GND	GND
DT	Data signal
SCK	Clock signal
V_{cc}	5V

Table 2.9: HX711 pins

2.3.6 Serial I^{2}C Module and 16*2 LCD

The liquid crystal display is a 16*2 screen commonly used in various devices and circuits. The LCD screen can show sixteen characters by two lines, each character is presented in a 5*7-pixel matrix. The display has two registers: Command Register and Data Register. The former stores the default command instructions such as initialization, cursor positioning, display control, etc. The latter stores the data to be shown on the LCD. The data are the ASCII values of the character to be shown, starting from 0 to 255. The robot is equipped with an LCD and an I^{2}C module that allows serial communication with it. This module is an adapter from the I^{2}C serial bus to the parallel bus used by the display and it has three connections named A1, A2 and A3 that have to be set as the module address, between 0x20 and 0x27. The hardware address must coincide with the software address. Our address is: 0x40.

Figure 2.11: LCD display and I^{2}C module

Pin	Function
GND	GND
V_{cc}	5V
SDA	Serial data bus
SCL	Serial clock bus

Table 2.10: I^{2}C module pins

2.3.7 Bluetooth Module

The HC-06 bluetooth module allows to use a UART/USART port, commonly known as a serial port, as a Bluetooth port. This device has a default baud rate of 9600 bit/s and it is used to communicate with the board via smartphone. It can only act as a slave device, in fact we start the communication at the beginning and never stop it. HC-06 uses “frequency hopping spread spectrum” technique (FHSS) to avoid interference with other devices and to have full duplex transmission, by the way in the project the communication from the MCU to the application is never used, it is only used a transmission in the opposite way.

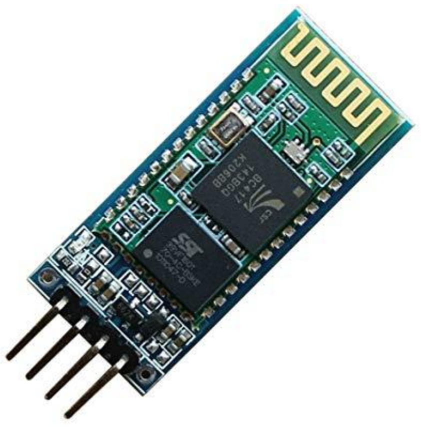


Figure 2.12: HC-06 Bluetooth module

Pin	Function
GND	GND
V_{cc}	3.3V
TX	Serial UART Tx bus
RX	Serial UART Rx bus

Table 2.11: Bluetooth module HC-06 pins

2.3.8 RGB LED

LED stands for Light Emitting Diode and it is a semiconductor device that emits light when an electric current is passed through it. RGB stands for Red, Green and Blue, the primary colors. The RGB LED has three diodes each used for one color. RGB Leds have four pins, however there are two kind of usage for them: Common Anode and three Cathodes that works with negative logic, Common Cathode and three Anodes that works with positive logic. This one is used in our project. Then through the combination of the three primary colors, it is possible to have many different ones, in particular in our project we used the three pins with the maximum brightness reaching 8 different colors.

Figure 2.13: LED RGB

Pin	Function
GND	GND
PB9	Blue
PA7	Green
PA9	Red

Table 2.12: Common Cathode configuration

2.3.9 Ultrasonic Sensor

The ultrasonic ranging module HC-SR04 provides 2 cm up to 400 cm non-contact measurement function and the ranging accuracy can reach 3 mm. The module includes ultrasonic transmitter, receiver and control circuit. The basic principle of work is that using IO trigger for at least 10 μs TTL high level signal, the module automatically sends eight 40 kHz pulses and detect whether there is a signal back. The signal back time of high output IO duration is the time from sending ultrasonic to returning.

Then comparing this with the speed of sound, it is possible to compute the distance used to know how far is the object from UOLL-I.

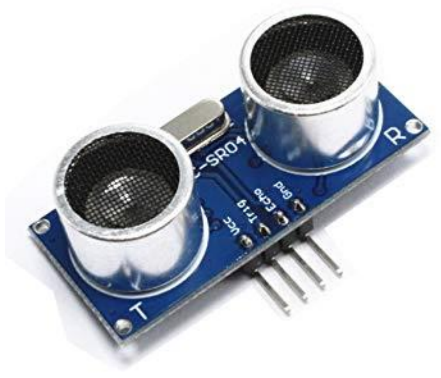
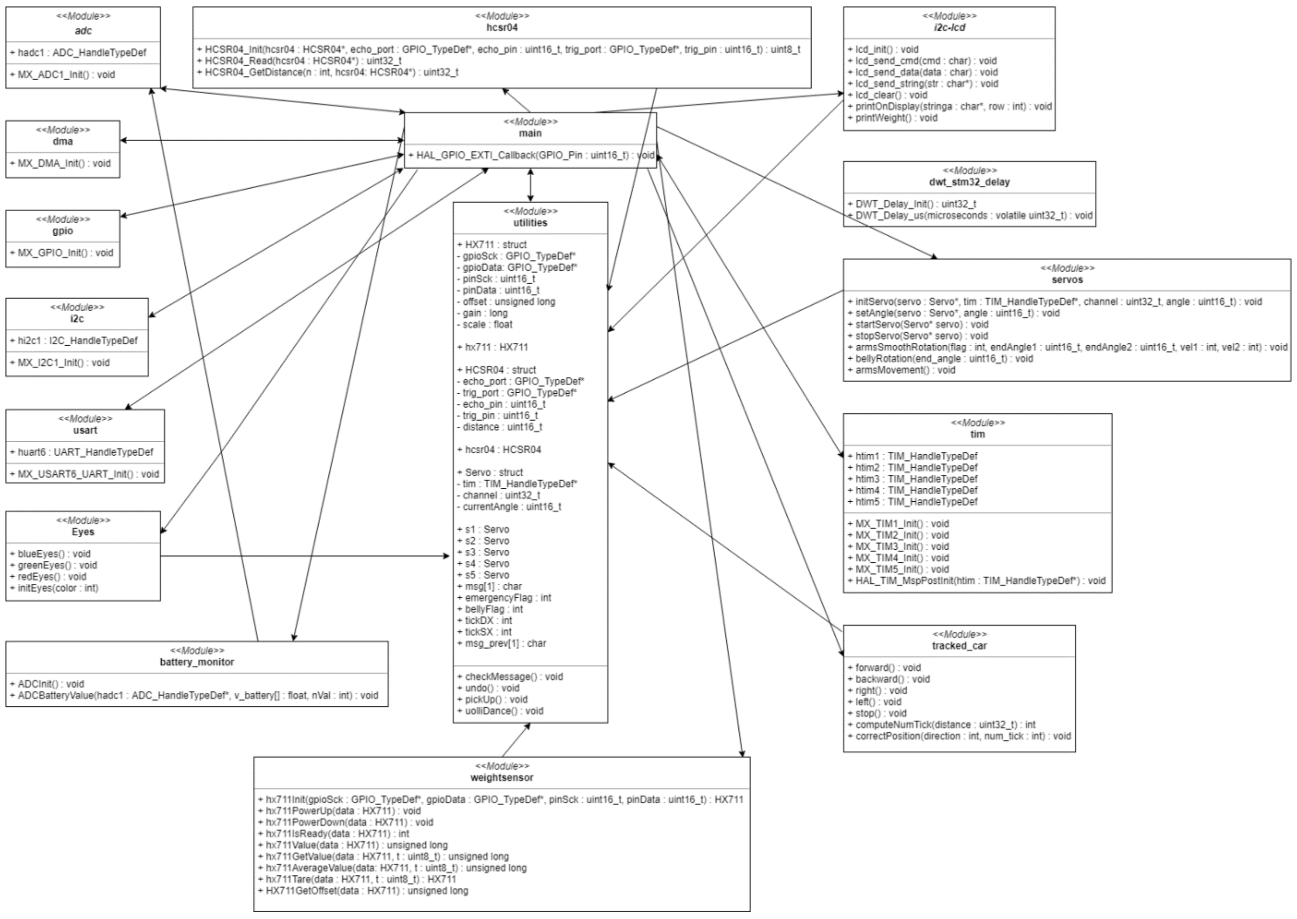


Figure 2.14: HC-SR04 Ultrasonic Sensor

Pin	Function
GND	GND
V_{cc}	5V
Trig	Trigger Pulse Input
Echo	Echo Pulse Output

Table 2.13: HC-SR04 pins

Chapter 3: Software Architecture



<<Module>>
adc
+ hadc1 : ADC_HandleTypeDef
+ MX_ADC1_Init() : void
<<Module>>
hcsr04
+ HCSR04_Init(hcsr04 : HCSR04*, echo_port : GPIO_TypeDef*, echo_pin : uint16_t, trig_port : GPIO_TypeDef*, trig_pin : uint16_t) : uint8_t
+ HCSR04_Read(hcsr04 : HCSR04*) : uint32_t
+ HCSR04_GetDistance(n : int, hcsr04: HCSR04*) : uint32_t
<<Module>>
i2c-lcd
+ lcd_init() : void
+ lcd_send_cmd(cmd : char) : void
+ lcd_send_data(data : char) : void
+ lcd_send_string(str : char*) : void
+ lcd_clear() : void
+ printOnDisplay(stringa : char*, row : int) : void
+ printWeight() : void
<<Module>>
dma
+ MX_DMA_Init() : void
<<Module>>
main
+ HAL_GPIO_EXTI_Callback(GPIO_Pin : uint16_t) : void
<<Module>>
dwt_stm32_delay
+ DWT_Delay_Init() : uint32_t
+ DWT_Delay_us(microseconds : volatile uint32_t) : void
<<Module>>
gpio
+ MX_GPIO_Init() : void
<<Module>>
utilities
+ HX711 : struct
- gpioSck : GPIO_TypeDef*
- gpioData: GPIO_TypeDef*
- pinSck : uint16_t
- pinData : uint16_t
- offset : unsigned long
- gain : long
- scale : float
+ hx711 : HX711
+ HCSR04 : struct
- echo_port : GPIO_TypeDef*
- trig_port : GPIO_TypeDef*
- echo_pin : uint16_t
- trig_pin : uint16_t
- distance : uint16_t
+ hcsr04 : HCSR04
+ Servo : struct
- tim : TIM_HandleTypeDef*
- channel : uint32_t
- currentAngle : uint16_t
+ s1 : Servo
+ s2 : Servo
+ s3 : Servo
+ s4 : Servo
+ s5 : Servo
+ msg[1] : char
+ emergencyFlag : int
+ bellyFlag : int
+ tickDX : int
+ tickSX : int
+ msg_prev[1] : char
+ checkMessage() : void
+ undo() : void
+ pickUp() : void
+ uolliDance() : void
<<Module>>
servos
+ initServo(servo : Servo*, tim : TIM_HandleTypeDef*, channel : uint32_t, angle : uint16_t) : void
+ setAngle(servo : Servo*, angle : uint16_t) : void
+ startServo(Servo* servo) : void
+ stopServo(Servo* servo) : void
+ armsSmoothRotation(flag : int, endAngle1 : uint16_t, endAngle2 : uint16_t, vel1 : int, vel2 : int) : void
+ bellyRotation(end_angle : uint16_t) : void
+ armsMovement() : void
<<Module>>
i2c
+ hi2c1 : I2C_HandleTypeDef
+ MX_I2C1_Init() : void
<<Module>>
usart
+ huart6 : UART_HandleTypeDef
+ MX_USART6_UART_Init() : void
<<Module>>
tim
+ htim1 : TIM_HandleTypeDef
+ htim2 : TIM_HandleTypeDef
+ htim3 : TIM_HandleTypeDef
+ htim4 : TIM_HandleTypeDef
+ htim5 : TIM_HandleTypeDef
+ MX_TIM1_Init() : void
+ MX_TIM2_Init() : void
+ MX_TIM3_Init() : void
+ MX_TIM4_Init() : void
+ MX_TIM5_Init() : void
+ HAL_TIM_MspPostInit(htim : TIM_HandleTypeDef*) : void
<<Module>>
Eyes
+ blueEyes() : void
+ greenEyes() : void
+ redEyes() : void
+ initEyes(color : int)
<<Module>>
battery_monitor
+ ADCInit() : void
+ ADCBatteryValue(hadc1 : ADC_HandleTypeDef*, v_battery[] : float, nVal : int) : void
<<Module>>
tracked_car
+ forward() : void
+ backward() : void
+ right() : void
+ left() : void
+ stop() : void
+ computeNumTick(distance : uint32_t) : int
+ correctPosition(direction : int, num_tick : int) : void
<<Module>>
weightsensor
+ hx711Init(gpioSck : GPIO_TypeDef*, gpioData : GPIO_TypeDef*, pinSck : uint16_t, pinData : uint16_t) : HX711
+ hx711PowerUp(data : HX711) : void
+ hx711PowerDown(data : HX711) : void
+ hx711IsReady(data : HX711) : int
+ hx711Value(data : HX711) : unsigned long
+ hx711GetValue(data : HX711, t : uint8_t) : unsigned long
+ hx711AverageValue(data: HX711, t : uint8_t) : unsigned long
+ hx711Tare(data : HX711, t : uint8_t) : HX711
+ HX711GetOffset(data : HX711) : unsigned long

Figura 3.1: Software Architecture.

It is used also a bluetooth module explained in the next paragraph that is not shown in figure.

3.1 Software Modules

The software modules created are shown below:

3.1.1 DMA Module

The DMA module has been designed to initialize DMA mode used by the Bluetooth module. Moreover, the transfer via DMA is used by the ADC module.

Bluetooth module has been used in DMA mode because it is one of the most used module, thus, through DMA, the CPU should not worry about that. Furthermore, it grants the minimum response time for "undo" operation, that is fundamental in emergency cases to avoid feasible damages to the robot.

3.1.2 GPIO Module

This software module has been designed to simplify the I/O management made available by the GPIO. Particularly, this module allows to select how to use a pin on the board.

3.1.3 I^2C Module

The I^2C module has been designed to manage the communication between the board and the LCD screen. This module provides basic functionalities for initializing the I^2C communication.

3.1.4 I2C-LCD Module

The module has been designed to use the basic features provided by the previous module and to implement simple and intuitive communication functions with the LCD screen.

- lcd_init (void): initializes the LCD display.
- lcd_send_cmd (char cmd): sends a command to the LCD.
- lcd_send_data (char data): sends a datum to the LCD.
- lcd_send_string (char *str): sends a string to be displayed to the LCD.
- lcd_clear (void): clears the LCD display.
- printOnDisplay(char* string, int row): displays a string on the LCD at a certain row.
- printWeight(void): computes and displays on the LCD the weight of the objects in the body.

3.1.5 USART Module

The USART module has been designed to manage the serial communication between the board and an external device. It has been used the USART6 to communicate via Bluetooth.

3.1.1 Bluetooth Module

The Bluetooth module has been designed to take advantage of the UART6 RxCpltCallback. This module provides the implementation of this callback in order to acquire the command sent by the user.

- HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart): default UART callback.

3.1.2 TIM Module

The TIM module initializes al the timers given by the board and used into the project. The timers used are shown below:

- TIM1: manages the belly servo motor
- TIM2: manages the movement of the left elbow servo motor
- TIM3: manages the movement of the left shoulder servo motor
- TIM4: manages the movement of the right elbow servo motor
- TIM5: manages the movement of the right shoulder servo motor

They are started and then stopped according to the operations that must be done.

3.1.3 Servos Module

The Servos module has been designed to initialize and simplify the management of the servo motor. The module provides some basic functionalities to move servos and some more complex one to manage the belly and the arms.

- initServo(Servo* servo, TIM_HandleTypeDef* tim, uint32_t channel,uint16_t angle): initializes the servo struct with the parameters given and starts the PWM signal.
- setAngle(Servo* servo, uint16_t angle): computes the PWM pulse, according to the minimum and the maximum ones, and sets the servo to the angle given.
- stopServo(Servo* servo): stops the PWM signal of the servo motor.
- startServo(Servo* servo): starts the PWM signal of the servo motor.
- bellyRotation(uint16_t end_angle): makes the servo motor of the belly move at a given angle.
- armsMovement(void): routine that makes both arms move according to the “pick up” sequence.
- armsSmoothRotation(int flag, uint16_t endAngle1, uint16_t endAngle2, int vel1, int vel2): function that moves the elbows servo motors to endAngle1 at vel1 speed and

the shoulder servo motors to endAngle2 at vel2. It is possible to choose the arm that has to be moved through the flag given: 0 for both, 1 for left, 2 for right.

3.1.4 ADC Module

The ADC module has been designed to initialize and manage the ADC. The ADC is used to read the battery voltage level in order to grant the correct execution of the command.

The ADC pin has a maximum input voltage of 3.6V, so to measure correctly the external battery voltage range of 0 - 9V, it is necessary to use a voltage divider.

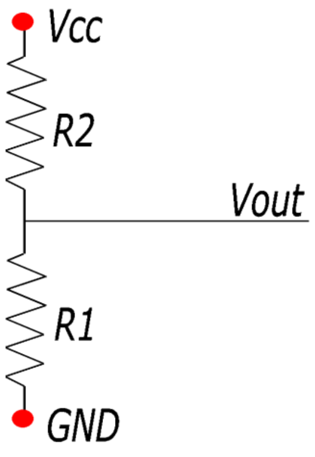


Figura 3.2: R1 = 2000 ohm, R2 = 3000 ohm for the first battery case

R1 = 2000 ohm, R2 = 2200 ohm for the second battery case.

3.1.5 Battery Monitor Module

The Battery Monitor Module has been designed to simplify the ADC management made available by the ADC module. In particular, this module allows to read the battery voltage level.

- ADCInit(void): initializes the ADC in DMA Mode.
- ADCBatteryValue(ADC_HandleTypeDef* hadc1, float v_battery[], int nVal): computes an average battery status given ‘nVal’ measurements to do.

3.1.6 DWT STM32 Delay Module

The DWT STM32 Delay module has been designed to manage the ultrasonic sensor microsecond delay. The module configures the DWT counter and provides a functionality to do a delay in microseconds.

- DWT_Delay_Init(void): initializes the DWT counter for the DWT Delay.
- DWT_Delay_us(volatile uint32_t microseconds): function that provides a delay in microseconds.

3.1.7 HCSR04 Module

The HCSR04 module has been designed to initialize and manage the ultrasonic sensor. The module provides functionalities to read the sensor and compute the object distance.

- HCSR04_Init(HCSR04* hcsr04, GPIO_TypeDef* echo_port, uint16_t echo_pin, GPIO_TypeDef* trig_port, uint16_t trig_pin): initializes the HC-SR04 struct.
- HCSR04_Read(HCSR04* hcsr04): function that starts a distance measurement.
- HCSR04_GetDistance(int n, HCSR04* hcsr04): gets the distance computed avoiding the ‘n/2’ smallest values and ‘n/2 -1’ greatest values in order to prevent uncorrect measurements.

3.1.8 Tracked Car Module

The Tracked Car module has been designed to simplify the management of the tracked car direction and the management of the encoders.

- forward(void): makes the robot to go forward.
- backward(void): makes the robot to go backward.
- right(void): makes the robot to go right.
- left(void): makes the robot to go left.
- stop(void): makes the robot to stop.
- computeNumTick(uint32_t distance): computes the number of ticks of the hall sensors to do from the given distance.

- correctPosition(int direction, int num_tick): adjust the current position moving in a direction (forward or backward) for a time interval related to the num_tick.

3.1.9 Weight Sensor Module

The Weight Sensor module has been designed to initialize and manage the weight sensor. The module provides functionalities to read the sensor and compute the object weight.

- hx711Init(GPIO_TypeDef* gpioSck, GPIO_TypeDef* gpioData,uint16_t pinSck, uint16_t pinData): initializes the weight struct.
- hx711PowerUp(HX711 data): switch on the weight sensor keeping the gpioSck set for 50 milliseconds and than reset it.
- hx711PowerDown(H;X711 data): switch off the weight sensor keeping the gpioSck set longer than 60 microseconds.
- hx711IsReady(HX711 data): checks if the weight sensor is ready.
- hx711Value(HX711 data): gets a single weight measurement.
- hx711AverageValue(HX711 data, uint8_t t): gets an average value.
- hx711Tare(HX711 data,uint8_t t): gets the tare value saving it into the struct.
- HX711GetOffset(HX711 data): gets the tare value.

3.1.10 Eyes Module

The Eye module has been designed to set the RGB LEDs to a certain color. Thus, the module provides functionalities as follow:

- blueEyes(void): sets the eyes to blue, used when UOLL-I is doing an operation.
- greenEyes(void): sets the eyes to green, used when UOLL-I is waiting for a command.
- redEyes(void): sets the eyes to red, used when UOLL-I is in “emergency”.
- initEyes(int color): sets the eyes to the color given by 8 possibilities.

3.1.11 Utilities Module

The Utilities module has been designed to simplify the management of the procedures and to save all the common structs and variables used. The module provides functionalities to manage these procedures:

- void messageCheck(void): checks the message received by the Bluetooth module and executes related operations.
- void undo(void): un-do the command in execution setting UOLL-I in emergency.
- void pickUp(void): procedure to pick up an object. It includes the movement toward the object, the opening of the belly, the weight measurement and come back to default position.
- void uolliDance(void): procedure to show different moves of each articulation of UOLL-I

About the common variables present in this module, the most important ones are:

- the struct of HX711 sensor and the HC-SR04 sensor one that contain different important values for the specific sensor, like the distance measured or the “Tare value”.
- The servo struct that allows to store the last setted angle useful for all the servos movements.
- The msg char value, it rappresents the command invoked by the user, it is setted during the DMA callback function and together with ‘msg_prev’ it gives information about user will
- The emergency flag is used to perform the “un-do” operation, avoiding to perform some undesired operation.