Indice

Capitolo 1

1.1 Phase 1

1.1.1 Hardware

- Quicrun 1060 brushed ESC.
  PWM frequency 50 Hz (200ms)
  It seems it wants 13 ms for neutral position. I tested with 10 as minimum value (maximum speed backward) and 20 as maximum (maximum speed forward).
  Pinout:
  - blue and yellow cables to the motor.
  - black and red connected to the power supply (battery). Be careful in not inverting them (black is the ground). The power supply should generate 7.2V and the current absorbed by the motor can also reach 2.5-3A.
  - The three small cables are connected to the driver. Black one is the ground, red one has to be connected to 5V and the white one to the PWM generator.
- Servo motor to drive the car direction.
  PWM frequency 50 Hz (200ms)
  It should work with 15 ms as central position, 20 and 10 ms for extremals. I tested it and it works but needs calibration yet (TODO).
- Rotative potentiometer. Watching it with pins in front of you:
  - Right pin on 3.3 V
  - Middle pin is the output (to the ADC)
  - Left pin on ground.
- STM32F4-discovery

  - TIM6: it drives the HighResolutionTimer and can be used as a synchronisation point for the execution cycle.
  - TIM4: drives the PWM generator. Now only the channel 1 (pin D12 connected also to green led) is activated.
  - ADC1: reads the speed reference. Now only the channel 8 is activated (pin B0). It acquires data transferring data using the DMA, but now after each conversion i put a sleep when i should poll on a flag asserting that the DMA has finished its work (TODO).
  - UART4: pins A0=Tx and A1=Rx (maybe is better another UART because the user button is connected to A0 and we want to use it (TODO)). It is used as the error stream flushing error messages.
  - USB-OTG-FS: uses the pins A11 and A12. This stream is used to get the RT diagnostic (much faster than UART).

1.1.2 Software

The code has been developed using MARTe framework libraries. A brief description of GAMs and DataSources involved in this first step follows.

- CoolCarControlGAM: reads the timer as synchronisation point and the ADC value from the potentiometer. The parameters that now can be defined are:
  - MaxMotorRef: the maximum value that can be read from ADC.
  - MinMotorRef: the minimum vale that can be read from ADC.
  - MaxMotorIn: the maximum PWM value (depending on the period configured on PWM timer)
  - MinMotorIn: the minimum PWM value.

  This GAM computes the PWM value linearly from the ADC respecting imposed ranges. Then writes on USB interface the value read from potentiometer and the PWM.

- TimerDataSource (input): Is the synchronisation point of the MARTe cycle. This DataSource supports only one signal specifying the sync frequency.
  - Frequency: is the parameter that has to be configured in the signal linked with this DataSource. A thread posts a semaphore with a frequency specified in the signal.

- USBSignalDisplay (output): It prints infos on USB interface.
  - FrameRate: specifies the print frequency.

A thread has been run printing all the values of the signals linked to this DataSource.

- PWM (output): Handles the PWM hw interface.
  - TimNumber: the timer handling the pwm signal.
  - Channel: the timer channel (max 4 for timer on STM32F4-Discovery)
  - ZeroVal: the value of 50% duty cycle (half period)

  Now each PWM DataSource can handle a single channel (TODO)

- ADC_DMA (input): Handles the read from ADC hw interface.
  - AdcNumber: the number of ADC used.
  - Channel: the ADC channel used (0-15 on STM32F4-Discovery).

  This DataSource manages the read from ADC using DMA to transfer data in memory.

The DataSources (and their brokers) normally don't manage the peripheral configuration. The configuration of all pheripherals has been generated using the STMCubeMX graphic tool. When opening the tool we must specify which type of board we are using. In this case we must specify (STM32F4-STM32F407/417-LQFP100) and then select the one with 1024 Kb flash and 192 Kb ram. The tool is quite intuitive for pheripheral configuration. To use the full speed clock we have to:

- Go to RCC in the Pinout area and select HSE with Crystal/Ceramic
- Go to the Clock Configuration area and set 8MHz in input frequency, select HSE and set M, N, P, Q to 8, 288, 2, 6 respectively in order to use ADC with the maximum speed (the max ADC speed is 72MHz then the clock speed has to be set to 144MHz which is the double).

It is possible configure all the peripherals paying attention to what we are using truth the MARTe DataSources. In general the DataSources manage only the pheripheral operations (read-write) and not the configuration, but if the pheripheral used by the DataSource has not be initialised that can cause unexpectable behaviors.
For this project we have to enable the TIM6 used as HighResolutionTimer, configure the TIM4 channel 1 for PWM output, the ADC1 channel 8 with the DMA2 stream 0, the USB in Device Only mode and Communication Device class and the UART4. We have also to enable FREERTOS. Once we have activated all the pheripherals in the Pinout area we have to switch in Configuration area to configure them as we wish. In FREERTOS configuration remember to enable USE_RECURSIVE_MUTEXES,

USE_RECURSIVE_MUTEXES and USE_COUNTING_SEMAPHORES. It is recommended using heap_3 as Memory Management scheme, so we don't have to care about the total heap size assigned to the operating system since we are using the standard heap memory area.
Once we have fully configured our board hardware go to Project-Settings and set the desired Project Name and Project Location and select Otger Toolchains in the Toolchain/IDE menu. Then going to Project-Generate code, all the configuration code will be generated in the specified folder.
The tool generates also an XML file with all the paths of the generated files. In the Scripts folder there is the python STM32ExtractFiles.py script which using the XML copies outside all the files.

- python STM32ExtractFiles.py [Project_Folder] [Project_Name] [Destination_Folder]

extracts the files from the folder generated by the tool to the destination folder specified. Use ToolConfiguration folder as destination folder remembering to delete all the *.c and *.h previous files but not other files such as the makefiles. The tool generates a main.c function which will be a little bit changed by the makefile. One of these changes is to add an include of a Addons.hdd files if it exists in that directory. Since all the pheripheral handles are declared statically in the main.c, in the Addons.hdd files we can define functions to get the handles and these functions will be used by the DataSources in order to manage pheripherals operations. The current Addons.hdd is:

```
UART_HandleTypeDef *errorUartHandle = &huart4;

ADC_HandleTypeDef * ADC_Handles[] = { &hadc1, NULL, NULL};

TIM_HandleTypeDef * TIM_Handles[] = { NULL, NULL, NULL, &
    htim4, NULL, &htim6, NULL, NULL };

TIM_HandleTypeDef * Get_TIM_Handle(unsigned int number) {
        return TIM_Handles[number];
}

ADC_HandleTypeDef * Get_ADC_Handle(unsigned int number) {
        return ADC_Handles[number];
}
```

where the defined function Get_TIM_Handle and Get_ADC_Handle are called by the DataSources.
The common STM32 DataSources are defined externally in the MARTe2-hw-testing where we are going to develop step by step the code that can be shared across different projects if they use the same type of hardware.
The makefiles manage all the compilation and linking mechanism across these different folders. The idea is to decouple the code that can be shared across different projects and the custom code of the single project we are developing. Acting on makefiles it

is possible decide the code to be compiled and linked. It is enough go in MARTe and call

- make -f Makefile.gcc

to build the project. The executable output.bin will be generated in MARTe/Build and it can be flashed on the board using programs such as st-link.