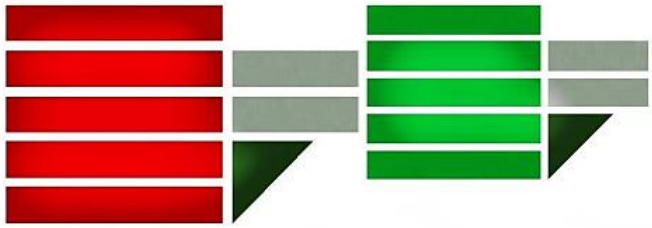
UNIVERSITÀDELLACALABRIA



Dipartimento di INFORMATICA , MODELLISTICA , ELETTRONICA e SISTEMISTICA

Master In Robotics and Automation

Walking Machine

Finale Project of Embedded system programming

Professor:
Prof. Gianni Cario

Student:
Alexis Marino-Salguero
(232601)

Academic year 202

Index

Introduction

The project consists of the construction of a scale system able to simulate the behaviour of an electric walking machine, this is fully based on the Embedded systems programming.

The main function involves a conveyor belt, which is able to reach different velocities according to the user, a motor to set up the inclination of the conveyor belt and input/outputs which give the control of the model. The microcontroller STM32F411RET6 has been used, this has an ARM Cortex-M architecture which is used exactly for embedded devices with low power consumption.

At the beginning is presented an introduction of logic under which the system will base its behaviour, the electronic elements used for the development the model and the reason for its choice.

Finally, the logic and form of programming used is explained according to the development of each step.

1. Objectives

The aim of this project consists of building a model and programming the microcontroller in order to construct a walking machine, the main objectives are:

- Generate controllers that allow us to configurate the velocity
- Generate an analogue input to set up the inclination of conveyor belt
- Configurate a 16x2 Lcd to show instructions and values.
- Configurate the DC gear motor and servo motor for the conveyor belt and inclination respectively
- Control the velocity of the conveyor belt with a controller
- Generate a code capable of follow the logic presented in fig.1.

The system's ignition control is through a switch, the speed control is carried out by two buttons that allow it to be lowered or raised. By means of the analogue reading of a potentiometer the angle of the conveyor belt is set. The motors are controlled by PWM signals, the LCD screen will show the speed and inclination of the conveyor belt as well as user interface messages.

The inputs, outputs and their configuration are shown in table 1. The code is developed in the stm32cubeIDE programming environment following the logic of the flow chart shown in fig.1.

Table 1. configuration of I/O ports

Name	Description	Pin	Configuration	Mode
Switch	On/off system	PA_10	Digital Input	Poling mode
Button 1	Increase velocity	PA_5	Digital Input	Interruption
Button 2	Decrease the velocity	PA_6	Digital Input	Interruption
Button 3	Emergency Stop	PA_7	Digital Input	Interruption
Pot 1	Inc/Dec angle	PA_0	Analog input	Interruption
Led red	Emergency signal	PC_7	Digital Output	
Led green	Normal work signal	PB_6	Digital Output	
Gear motor	Conveyor belt	PB_5	Alternate Function	
Servo motor	Inclination	PB_4	Alternate Function	
Encoder	Read the velocity	PC_2	Digital Input	

Lcd-D4	Lcd configuration	PA_13	Digital Input	
Lcd-D5	Lcd configuration	PA_14	Digital Input	
Lcd-D6	Lcd configuration	PA_15	Digital Input	
Lcd-D7	Lcd configuration	PC_13	Digital Input	
Lcd-RS	Lcd configuration	PC_10	Digital Input	
Lcd-EN	Lcd configuration	PC_12	Digital Input	

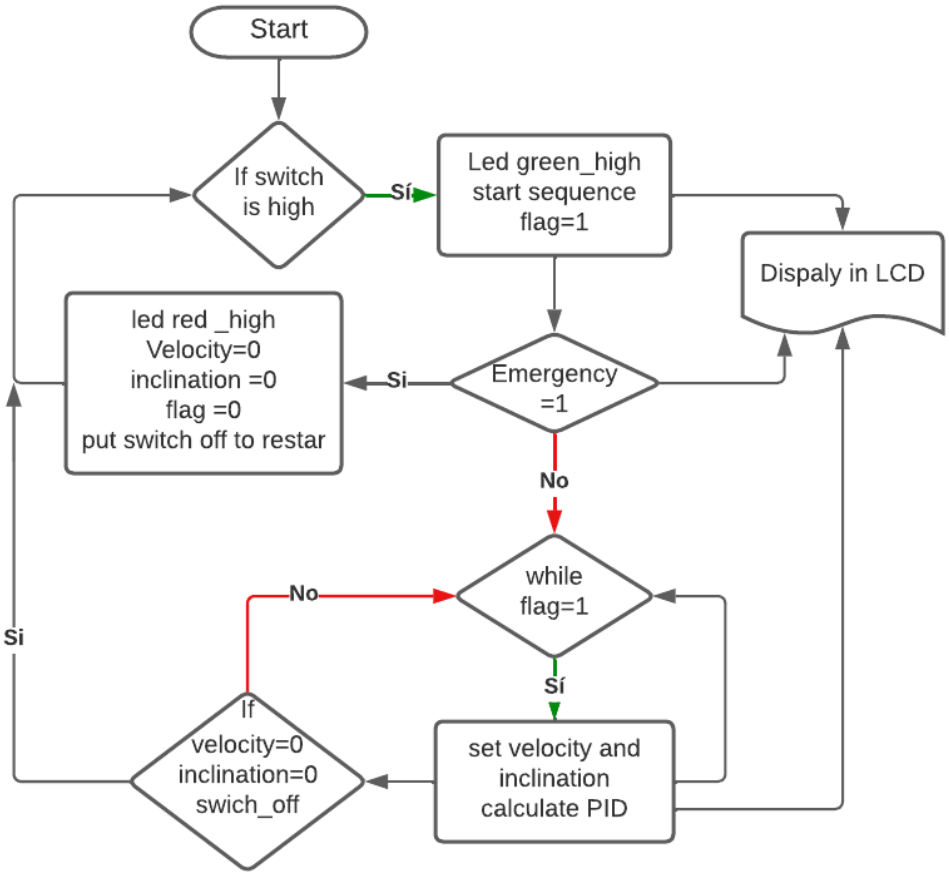


Fig.1 Programming logic Flow chart

2. Controls for the systems “GPIO module”

The General-Purpose Input/Output peripheral is a hardware interface used to allow the board to interact with the external environment.

2.1. Switch (on/off the system)

Fig.2 swith

this device maintains the state high or low depending on where it is located. the comparison of this state is done in the while true loop and allows the entire system to be activated. At first it was thought to use this signal in interrupt mode to prevent it from being read constantly. Although, since the device maintains the level, the interrupt had to be activated to detect rising and falling edges. The problem focuses on the fact that, as it is a mechanical element, it produces too many bounces, phenomenon shown in Fig. 3, so the interruption occurs more times than necessary. however, a low-pass RC filter was applied to reduce this unwanted behaviour, but the interruption did not work correctly. So that, it is proposed for future versions to apply a filter by software to reduce this phenomenon and configure the switch in interruption mode.

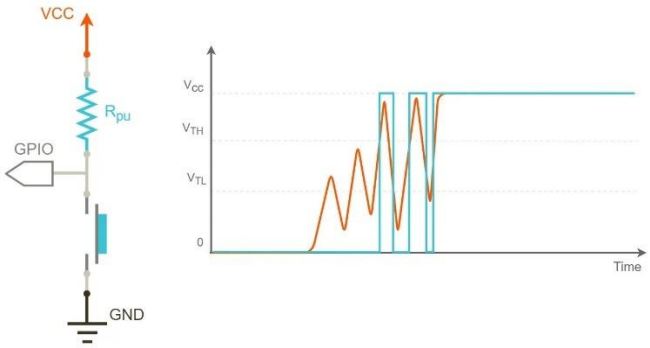


Fig.3 bounces on pushbuttons

- *Programming, configuration of registers for PA_10 input:*

First, it is necessary activate the clock for GPIOx, how the project uses pins in the ports A,B and C, the three clocks are activated by setting this register.

```
Enable clock on GPIOA/GPIOB/GPIO
RCC->AHB1ENR |= 0x07;
```

The configuration of the registers to configurate a input is:

```
GPIOA->MODER &= ~(0x03 << 20); // Clear MODER GPIOA10
GPIOA->MODER |= (0x00 << 20);  // Set Input
GPIOA->PUPDR &= ~(0x03 << 20); // Clear PUPDR GPIOA10
GPIOA->PUPDR |= (0x00 << 20);  // Set Pull-Up
```

2.2. Buttons (increase and decrease the velocity and emergency stop)

Two ordinarily open push buttons perform the increase and decrease of velocity, other is used as stop emergency that simulate the situation when an accident occurs in the walking machine. These are configured in pull-up way so that when these are pressed send ground to the board.

Fig.4. push button

The operation of these is in Interrupt mode because reading these ports is only required if the buttons are pressed. The logic for increase or decrease the velocity is as follows, each time that interruption occurs a flag increase or decrease its value, then this flag is used in the While true to set the velocity. For the emergency stop, the interrupt occurs and set a flag in 1.

- *Programming, configuration of registers for PA_5, PA_6 and PA_7 input:*

The configuration of the pins as inputs is equal to PA_10, but with the number of pins respectively.

The configuration of the interruption is done as follows.

First enable the clock for interruptions

```
// Enable the clock for SYSCFG (bit 14)
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

Configurate the register for interruptions.

```
SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI6_PA; // External interrupt on GPIOA6
SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI5_PA; // External interrupt on GPIOA5
SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI7_PA; // External interrupt on GPIOA7
SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PC; // External interrupt on GPIOC2

EXTI->IMR |= (0x01 << 6) | (0x01 << 5)| (0x01 << 7)| (0x01 << 2);     // Set not masked interrupt
EXTI->RTSR |= (0x01 << 7);    // Rising Edge
EXTI->FTSR |= (0x01 << 6) | (0x01 << 5)| (0x01 << 2);    // Falling Edge
```

SYSCFG is an external interrupt configuration, this configurate the pin and the GPIO port that is used, also it is configurated in the interruption is performed by falling or rising edge. Note that PC_2 is also configured, this interrupt will be used later.

Manage of NVIC

```
/* Gestione NVIC */
// PA_5 Y 6 y 10
NVIC_SetPriority(EXTI9_5_IRQn, 0);
NVIC_ClearPendingIRQ(EXTI9_5_IRQn);
NVIC_EnableIRQ(EXTI9_5_IRQn);
// Abilitazione Interrupt
__asm volatile ("cpsie i" : : : "memory"); // Change Processor State, Enable Interrupts
```

Finally, the function Handler that is call went the interruption occurs

```
void EXTI9_5_IRQHandler(void){
  if(EXTI->PR & (0x01 << 5)){
    NVIC_ClearPendingIRQ(EXTI9_5_IRQn);
    if(velocity<3)
    velocity++;
    else if (velocity>=3)
        velocity=3;
    EXTI->PR |= (0x01 << 5); // Clear the EXTI pending register
  }
  if(EXTI->PR & (0x01 << 6)){
      NVIC_ClearPendingIRQ(EXTI9_5_IRQn);
      if(velocity>0)
         velocity--;
         else if (velocity<1)
            velocity = 0;
      EXTI->PR |= (0x01 << 6); // Clear the EXTI pending register
    }
  if(EXTI->PR & (0x01 << 7)){
      NVIC_ClearPendingIRQ(EXTI9_5_IRQn);
      emergency = 1;
      EXTI->PR |= (0x01 << 6); // Clear the EXTI pending register
    }
}
```

It is important notice that, how the pins 5,6 and 7 are used. It is necessary put conditions in order to identified from which input is coming the interruption because these are joined in a single external interrupt request EXTI9_5_IRQ.

2.3. Potentiometer (set the inclination)

A potentiometer was chosen for inclination adjustment in order to use the microcontroller's ADC.

Fig.5 potenciometer

This input works in interrupt mode because this value only needs to be measured when the user wants to do changes.

- *Programming, configuration of registers for PA_0 as analog input:*

To configure the ADC was use the tools provided by STM32CubeMX, configuring channel 0 of the ADC as shown in the fig.6.

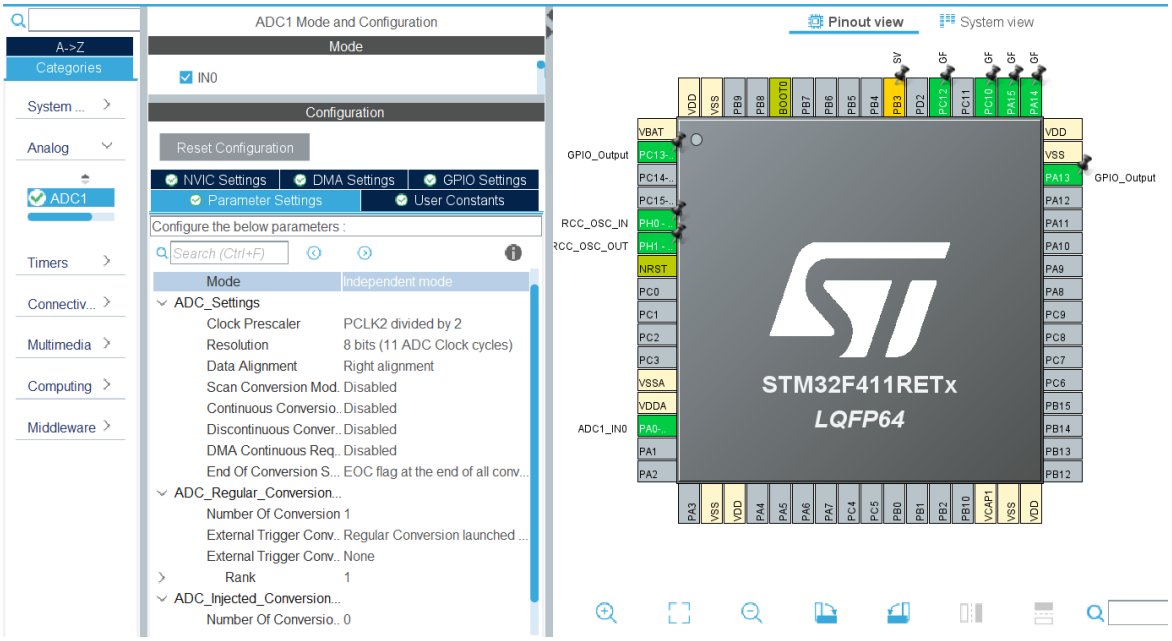


Fig.6. ADC configuration

The system clock is working at 60Mhz, the Prescaler chosen was 2 then the register APB2 which has the ADC clock is configured at 30Mhz. the resolution was 8 bits because the idea is move the conveyor belt from 0 to 45 degrees and in terms of duty cycle of PWM it means values from 0 to 2.5, it will be shown later, then with 8 bits the ADC is able to differentiate 256 values. The sampling time is 3 cycles. These set the conversion time as:

$$Tconv = \frac{sampling\ time + 8}{adc\ clock}$$

$$Tconv = \frac{3 + 8}{30\ Mhz} = 0.3us$$

The continues conversion mode is disable because this will be set up as interruption in this way

```
// Interrupt configuration for ADC
NVIC_SetPriority(ADC_IRQn, 0);
NVIC_EnableIRQ(ADC_IRQn);
// Start ADC in interrupt Mode
HAL_ADC_Start_IT(&hadc1);
```

Enable the interruption for ADC, ISR specifications are used to control the conversion of the ADC, the functions that manage interrupt requests are

```
void ADC_IRQHandler(void) {
    HAL_ADC_IRQHandler(&hadc1);
    // Re-Start ADC in interrupt Mode
    HAL_ADC_Start_IT(&hadc1) ;
}
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    // Read the analog input value
    raw_in = HAL_ADC_GetValue(&hadc1);
    // Converts value in the 0V-3.3V range
    Pot_mv = (((float)raw_in) /256 ) * 2.5;
    // Convert mV to angle
    Potshow=(Pot_mv*45)/2.5;
}
```

The first handles interrupt requests, the second is automatically invoked when a conversion ends and change the potentiometer measurements so that the data is sent to PWM and the angle displayed on the LCD.

2.4. Led (normal and emergency signals)

2 leds were placed to show if the system is working correctly and when there is an emergency stop.

- *Programming, configuration of registers for PB_6, PC_7 as output:*

```
/** GPIOB Configuration **/
GPIOB->MODER  &= ~(0x03 << 12); // Clear GPIOB6
GPIOB->MODER |= (0x01 << 12);  // Set Output
// Push-Pull output & Pull-up
GPIOB->OTYPER &= ~(0x1 << 6);
GPIOB->PUPDR &= ~(0x03 << 12); // Clear GPIOB6
GPIOB->PUPDR |= (0x01 << 12);
```

Notice that the register OTYPER is a register that only can be configurate 16 bits the other register has 32 bits to configurate two for each, also is important mention that if the register OSPEEDR is not configurate it take a default value that is 00 “low speed”.

3. Principal actuators

the main actuators are a geared motor for the band and a servo motor for the tilt angle

3.1. Gear motor

In the project, the motor that have been used are - more specifically - DC gear motors, with a gearbox ratio equal to 1:48; this means that the internal DC motor's velocity is reduced by a factor of 48 the maximum velocity without load is 170 RPM.

Fig.7. gear motor

The motor is controlled whit a H-bridge (L293D), this h-bridge is an electronic component that allows to drive two dc motors in both of direction, the chip has 16 pins:

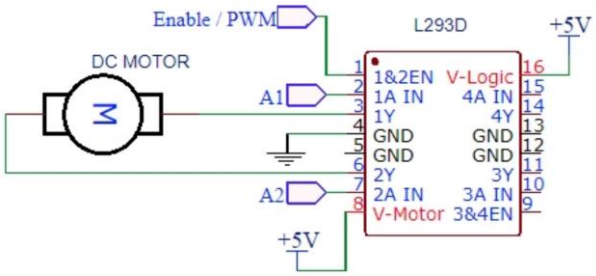


Fig.8 L293D h-bridge

How the belt goes only in one direction the pins A1 and A2, which control the direction of rotation of the motor, are set one in high and the other in low.

In order to control the motor, an PWM signal is necessary. The timer used was configured as follows:

```
// GEARMOTOR
GPIOB->MODER &= ~(0x03 << 10);      // Clear GPIOB5
GPIOB->MODER |= (0x02 << 10);       // Alternate Function (AF) Mode
GPIOB->PUPDR &= ~(0x03 << 10);
GPIOB->PUPDR |= (0x01 << 10);       // Pull-up
GPIOB->AFR[0] |= (0x02 << 20);      // [0]=AFRL, <<20 = pin5, 0x02 = AF2
// TIMER 3 CONFIGURATION
RCC->APB1ENR |= (0x01 << 1);        // clock TIM3
TIM3->PSC = 5999;
TIM3->ARR = 200;
// ch2
TIM3->CCMR1 |= (0x6 << 12);         // PWM Mode in Capture/Compare register (ch2)
TIM3->CCER |= (1 << 4);             // Enable Capture/Compare output (ch2)
TIM3->CR1 |= (1 << 0);              // Counter Enable
```

The timer used was the 3 and the channel two its own pin are connected to the h-bridge. It is important to make sure that the frequency of the PWM signal is high enough, in this case a high frequency is kept in order to avoid problems with the motor.

$$f = \frac{60000000}{(5999+1) * (199+1)} = 20ms$$

In order to setup the duty cycle the following formula is applied

$$Dcycle = \frac{CCRx}{AARx}$$

It means that the duty cycle from 0 to 100% correspond to values from 0 to 200,For example

```
TIM3->CCR2 &= 0;
TIM3->CCR2 = 0;                    // Duty-cycle (ch2)
```

This is the fist configuration to start the motor without movement.

Parameters of the motor

R=0.079

L=0.016

K=0.079

J=0.0062

B=0.00087

Mechanical equation of the motor

$$\frac{w(s)}{v(s)} = \frac{K_m}{js + B}$$

$$f_{PWM} \geq \frac{5}{2*\pi*\tau} \quad , \quad \frac{w(s)}{v(s)} = \frac{k}{\tau s+1}$$

3.2. Servo motor

The servomotor used is the Tower Pro, model SG90: a small servomotor that can rotate the axis about 180 degrees, but not able to develop a significant torque (1.80 Kg/cm). It required of a PWM signal that has a frequency of 50Hz and a variable duty cycle.

Fig.9 servo motor

The datasheet of the stepper motor specified that the minimum angular position is (0°) called Minimum_Pulse and the pulse width that corresponds to the maximum angular position (180°) and its call this the Maximum_Pulse.

duty cycle 1ms: 0 degrees

duty cycle 2ms: 180 degrees

The timer used to control this motor is the same timer 3 but now using the channel 1, the period is 20ms, the servo motor has a minimum pulse width of 1ms (for the 0° position) and a maximum pulse of 2ms (for the 180° position). This means that the entire range of PWM signal that covers the full motor rotation is from 5% up to 10%, which means that the structure in the CCR1 register will have to vary approximately between 10 and 20.

The idea is setup the inclination from 0 to 45 degrees, then the maximum value for the register CCR1 will be 12.5 its answer why the configuration of potentiometer compute values from 0 to 2.5. The register configuration for servo motor is:

```
// MICRO SERVO
 GPIOB->MODER &= ~(0x03 << 8);       // Clear GPIOB4
 GPIOB->MODER |= (0x02 << 8);        // Alternate Function (AF) Mode
 GPIOB->PUPDR &= ~(0x03 << 8);
 GPIOB->PUPDR |= (0x01 << 8);        // Pull-up
 GPIOB->AFR[0] |= (0x02 << 16);      // [0]=AFRL, <<16 = pin4, 0x02 = AF2
 // TIMER 3 CONFIGURATION
 // ch1
 TIM3->CCMR1 |= (0x6 << 4);          // PWM Mode in Capture/Compare register (ch1)
 TIM3->CCER |= (1 << 0);             // Enable Capture/Compare output (ch1)
 TIM3->CR1 |= (1 << 0);              // Counter Enable
 TIM3->CCR1 &= 0;
 TIM3->CCR1 = 10;                    // Duty-cycle (ch1)
```

4. Control of the gear motor

In order to perform a control of the motor its necessary to have a dispositive able to measure the motor output in this case the velocity. This goal is achieved with an encoder.

4.1. Encoder (Photo Interrupter Module)

It applies the principle that light is interrupted when an object passes through the sensor, so it's a switch that will trigger a signal when light between its gap is blocked.
Therefore, photo-interrupters are widely used in many applications, like speed measurement.

Fig.10. Encoder

The pins are three: Vdd, GND and a digital output.

The operating principle refers to a pull-up configuration because the sensor when a hole is detected, the digital output becomes high, otherwise it's equal to zero; this is the reason why the rising edges are considered.

The logic that has been implemented involves the use of a counter, in interrupt mode, that is incremented at each rising edge of the digital output. In fact, a wheel with 20 holes is in the sensor's gap and spins at a certain velocity that we want to measure

If the measurement process is performed in a minute, we are able to get the number of rpm

$$rpm = \frac{count}{20} * 60$$

Max velocity whit 9 volts 175 rpm.

- *Programing and configuration of register*

```
/* Timer 1 to check the encoder */
   RCC->APB1ENR |= (0x01 << 0);    // Clock TIM2
   TIM2->PSC = 5999;
   TIM2->ARR = 3999;
   TIM2->DIER &= ~0x01;
   TIM2->DIER|=0x01;                  // Interrupt Enable
   NVIC_ClearPendingIRQ(TIM2_IRQn);
   NVIC_EnableIRQ(TIM2_IRQn);
   //__enable_irq();
   TIM2->CR1 |= (0x01 << 0);
```

The starting frequency is equal to 60MHz. The interrupt was used to store the counter´s value in a variable each 0.4 seconds. To process it in an adequate way, with the aim to compute the wheel's speed. The ISR (Interrupt Service Routine) has been defined in this way:

```
void TIM2_IRQHandler(void){                    // Encoder check
    NVIC_ClearPendingIRQ(TIM2_IRQn);
    TIM2->SR= (uint16_t)(~(1 << 0));
    c=c_p;                                     // storing the number of pulses
    c_p = 0;                                   // pulses counter reset
    pid=1;
}
```

Let's focus just on the counters' values storing and reset. As shown in the follow picture if an interrupt occurs, the value of the counter increase.

The variable pid set to one in the interrupt service routine of TIMER2 each 0.4 second allows to use PI controller each sampling time. For do it that variable is checked inside the while true after choosing the reference to track.

```
void EXTI2_IRQHandler(void){
            // GPIOC2
    NVIC_ClearPendingIRQ(EXTI2_IRQn);
    c_p++;

    EXTI->PR |= (0x01 << 2);                   // Clear the EXTI pending register
}
```

Another ISR concerns the encoder's digital output; as it has been shown in the previous section, it's important to consider its falling edges in order to increment the counter's value: on the basis of this purpose, the interrupt on the rising edges has been activated. The PIN selected is PC2 and the configuration of this interruption was shown before.

4.2. Controller PI

Due to the fact that a dc motor has first order dynamics, a PI controller is enough to correct the error in space state and have an adequate system response.

with proportional action we try to minimize the system error. When the error is large, the control action is large and tends to minimize this error. while the integral action is achieved to reduce the error of the system in permanent regime.

In order the to follow the different reference of speed a PI control was introduced. The speed references are three: 50, 75, 110 rpm.

due to not having the dynamics of the plant the Tuning of the PI was done manually. The proportional and integrator gains was chosen by empirical way with this purpose. The proportional gain chosen was $k_p = 0.5$ and the integral gain was $k_i = 4$.

Is important to say the sampling time used is $T_s = 0.4$ seconds, the reason to have a long time is that allow to have more accuracy in rpm measure which is

fundamental to able the motor to follow the different reference signals.

```
int calculate_PID(){

    float error = ref-c;
    integral += (4*0.4*error);                      //ki=4, ts=0.4
    float u_input = (0.5*error)+integral;           // strategia PI  kp=0.5
    //u_input = anti_windup(u_input, err, sPID);
    if (u_input > u_max){
        u_input = u_max;
        integral-=(4*0.4*error);
    }
    else if (u_input < u_min){
        u_input = u_min;
        integral-=(4*0.4*error);
    }

return (int)u_input;


}
```

The anti-windup filter was used to bound the output provided by the controller to avoid some motor´s velocity not desired in the system.

5. LCD

Basically, the parallel connection between STM32, and the LCD was used. LCD 16×2 can be connected in the parallel mode using 4 data pins (LCD 4 bit MODE).

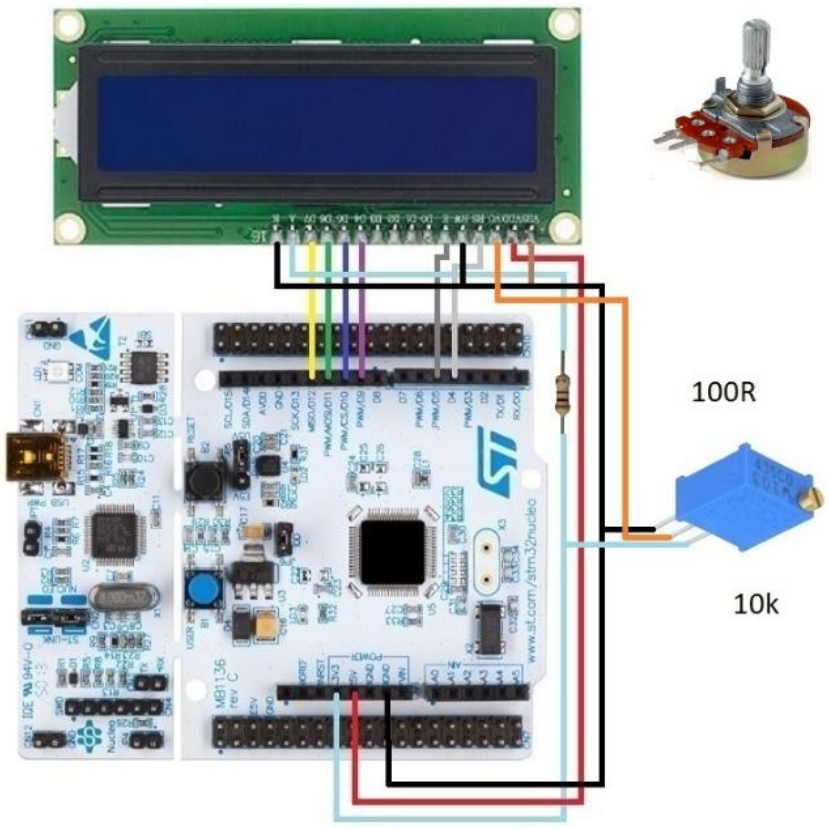


Fig.11 Connection of LCD and micro

The picture above show the connection of the lcd, in this project the connection is, the higher 4 pins of the LCD (D4-D7) are connected to the microcontroller pins (PA13-PA14-PA15-PC13). And the others pins.

RS is connected to PC10

RW is connected to Potentiometer

EN is connected to PC12

The pins use as output were configurated in the graphical interface of IDE

```
// Lcd_PortType ports[] = { D4_GPIO_Port, D5_GPIO_Port, D6_GPIO_Port, D7_GPIO_Port };
  Lcd_PortType ports[] = { GPIOA, GPIOA, GPIOA, GPIOC };
  // Lcd_PinType pins[] = {D4_Pin, D5_Pin, D6_Pin, D7_Pin};
  Lcd_PinType pins[] = {GPIO_PIN_13, GPIO_PIN_14, GPIO_PIN_15, GPIO_PIN_13};
  //Lcd_HandleTypeDef lcd;
  // Lcd_create(ports, pins, RS_GPIO_Port, RS_Pin, EN_GPIO_Port, EN_Pin, LCD_4_BIT_MODE);
  lcd = Lcd_create(ports, pins, GPIOC, GPIO_PIN_10, GPIOC, GPIO_PIN_12, LCD_4_BIT_MODE);
  Lcd_clear(&lcd);
```

A library that manages the use of the lcd was taken from an ATM32 programming forum, which among its main functions has.

```
void Lcd_init(Lcd_HandleTypeDef * lcd);
void Lcd_int(Lcd_HandleTypeDef * lcd, int number);
void Lcd_string(Lcd_HandleTypeDef * lcd, char * string);
void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col);
Lcd_HandleTypeDef Lcd_create(
        Lcd_PortType port[], Lcd_PinType pin[],
        Lcd_PortType rs_port, Lcd_PinType rs_pin,
        Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode);
void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]);
void Lcd_clear(Lcd_HandleTypeDef * lcd);
```

Where:

Lcd_int: allow to display a integer in the lcd.

Lcd_string: allow to display a chain of characters in the lcd.

Lcd_cursor: put the cursor of the LCD in the desired position.

Lcd_clear: clean the display of the lcd

6. WHILE TRUE

Finally, the programming logic that follows the configurations shown in fig.1 is shown in the while true

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    if (!(GPIOA->IDR >> 10 & 0x1) & z==0){
         GPIOB->ODR  |=  (0x01 << 6);
         GPIOC->ODR  &= ~  (0x01 << 7);
         x=0;
         start();
        while (y==1){
             if(emergency == 1){
               Lcd_clear(&lcd);
               Lcd_cursor(&lcd, 0,3);
               Lcd_string(&lcd, "EMERGENCY");
               Lcd_cursor(&lcd, 1,6);
               Lcd_string(&lcd, "STOP");
               EmergencyStop();
               Lcd_clear(&lcd);
              }
             velocities();
           if(pid==1 && velocity>0){
               pwm=calculate_PID();
              TIM3->CCR2 &= 0;
              TIM3->CCR2 = pwm;
           }
           pid=0;
             Lcd_cursor(&lcd, 0,0);
             Lcd_string(&lcd, "Velocity:");
             Lcd_cursor(&lcd, 0,9);
             Lcd_int(&lcd, velocity);
             Lcd_cursor(&lcd, 0,13);
             Lcd_int(&lcd, ref);
             if (ref<10){
                 Lcd_cursor(&lcd, 0,14);
                 Lcd_string(&lcd, " ");
             }
              if (ref<100){
                Lcd_cursor(&lcd, 0,15);
                Lcd_string(&lcd, " ");
              }
```

```
            Lcd_cursor(&lcd, 1,0);
            Lcd_string(&lcd, "Inclination:");
            if (Pot<10){
                Lcd_cursor(&lcd, 1,14);
                Lcd_string(&lcd, " ");
            }
            Lcd_cursor(&lcd, 1,13);
            Lcd_int(&lcd, Potshow);
            //velocities();
            TIM3->CCR1 &= 0;
            TIM3->CCR1 = 50+Potshow;
            if ((GPIOA->IDR >> 10 & 0x1) & Pot<5 & velocity==0){
                z=1;
                y=0;
            }
        }
    }
    else if((GPIOA->IDR >> 10 & 0x1) & z==1){
          GPIOB->ODR  &= ~  (0x01 << 6);
          GPIOC->ODR  &= ~  (0x01 << 7);
          Lcd_clear(&lcd);
          while  (x<=5){
              Lcd_cursor(&lcd, 0,4);
              Lcd_string(&lcd, "GOODBYE");
              HAL_Delay (650);
              Lcd_clear(&lcd);
              GPIOC->ODR ^= (0x01 << 7);
              GPIOB->ODR ^= (0x1 << 6);
              HAL_Delay (650);
              x++;
          }
          GPIOB->ODR  &= ~  (0x01 << 6);
          GPIOC->ODR  &= ~  (0x01 << 7);
          z=0;
          x=0;
          Lcd_clear(&lcd);


        }
        if (!(GPIOA->IDR >> 10 & 0x1) & z==1){

              GPIOC->ODR  &= ~  (0x01 << 7);
              Lcd_cursor(&lcd, 0,0);
              Lcd_string(&lcd, "PUT SWICH TO OFF");
              Lcd_cursor(&lcd, 1,0);
              Lcd_string(&lcd, "    TO RESET    ");
               TIM3->CCR2 &= 0;
               TIM3->CCR2 = 0;
               TIM3->CCR1 &= 0;
               TIM3->CCR1 = 50;
        }

    }
    /* USER CODE END 3 */
}
```