ECOLE
POLYTECHNIQUE
DE BRUXELLES

Université Libre de Bruxelles

ELEC-H309 - Integrated Project

Integrated Project's Report

Author:
Alexandre Boving
Serge Agyemang

Code Number:
000459850
000494640

Supervisor:
Michel Osée

IRCI 3 - Electronics and Telecommunication Module

20th of May 2022

Contents

1 Introduction **4**

2 Project Analysis **5**

2.1 Document Requirements 5

2.2 Functional Modules 6

2.3 Validation Strategy 6

3 Audio Communication **7**

3.1 Audio Acquisition Chain 7

3.1.1 Frame Command 7

3.1.2 Analog circuit 8

3.1.3 Validation of the audio acquisition chain 11

3.2 Digital signal processing 12

3.2.1 Digital filters 12

3.2.2 The comparator and the FSK detector 14

3.2.3 Exchange of information between the *dsPIC* via *UART* 15

3.2.4 Validation of the acquisition audio channel 15

4 Vehicle movement **16**

4.1 Description of the main components 16

4.2 Polar regulation 16

4.2.1 Set-point Generation 17

4.2.2 Proportional Controller 19

4.2.3 Encoders and Feedback 19

4.3 Validation de la partie déplacement 21

4.3.1 Traitement de la zone morte et précision 21

5 Conclusion **22**

6 Appendix **23**

6.1 GitHub Link 23

6.2 Robot movement (second variant) 23

1 Introduction

The Integrated Project course involves designing a control system for a robot capable of movement based on a command transmitted in the form of a bitstream. This bitstream exists as an audio signal, achieved through Frequency Shift Keying **FSK** modulation, before being processed by the robot's audio communication channel. Essentially, this control system must enable our automated vehicle to process audio command signals, allowing it to execute precise movements autonomously. The robot is expected to follow received commands such as 'Move forward,' 'Move backward,' 'Turn left,' and 'Turn right,' accompanied by a parameter indicating the distance or angle to be covered.

To accomplish this task, the overall work has been divided into two parts. The first part deals with the acquisition of the audio signal and its digital processing, referred to as 'Audio Communication.' The other part primarily focuses on the control loop implemented within the robot, managing its movement, termed 'Vehicle Movement.' The connection between these two parts is established to achieve the complete control system. Additionally, comments related to our code will be provided at the end of the report for further clarity. Regarding our codes, they are accessible via a GitHub link and are accompanied by our research notes.

2 Project Analysis

Before delving into the first part of our report concerning audio communication, a critical project analysis section is necessary to thoroughly grasp the project and establish a working methodology.

2.1 Document Requirements

The specifications are provided in the document 'Project Analysis' located in the UV folder. However, for a more comprehensive analysis of the problem, the specifications are reiterated below.

1. Functions
    - Behave entirely autonomously.
    - Receive and treat audio command signals.
    - Execute the following orders: "**Forward**", "**Backwards**", "**Turn Left**", "**Turn Right**".
2. Field Constraints
    - Robot must be able to move on a solid, steady and horizontal terrain.
    - Robot must be able to function in a relative noise-quiet environment.
3. User Constraints
    - Power tension must be under/or equal to 24V threshold.
    - Translation speed of the robot must be under 1m/s.
    - Robot must be compact enough to let the students pick it up and insert it in a bag.

From the Functions section of the specifications, it is evident that the project can be directly divided into two parts, as mentioned in the introduction. Receiving and understanding the command signals can be handled independently of the robot's physical actions following the received orders. In the Environmental Constraints section, the field assumptions within which the robot operates are specified. The term 'quiet' environment provides an indication of the possible test(s) to be conducted to meet this constraint. An experiment conducted involves adjusting the sensitivity of the robot's audio command signal detection to identify the bitstream

under noisy circumstances, such as background music. Consequently, a calm environment should not pose significant issues. The final category of constraints, User Constraints, is adhered to from the project's outset since the power supply voltage (7V, to be verified), translation speed (0.5 m/s), and robot geometry are directly provided.

2.2 Functional Modules

Within the two main sections of this report, several modules can be individually isolated to verify their validity. In this case, this subsection serves to establish an existing working methodology among many others. The team of engineers responsible for this project has chosen to work according to the **Agile** method. By setting short-term objectives, it is easier to break down the project into several subcategories that must be gradually achieved. These modules, which we attempt to conclude and validate at the end of one or two sessions, are the goals of what is referred to as a 'sprint.' In total, several 'sprints' have been conducted for the following modules: **Amplification Circuit | Guard Filter**, **Digital Filters**, **Comparator/FSK Detector**, **Motor Start-up**, **Regulation Loop**, and **Communication between the two dsPIC units**.

2.3 Validation Strategy

The validation stage through experiments represents a significant advancement in the project, as it allows for the closure and validation of a module before integrating it into the entire robot system. Throughout this report, the modules are presented along with the accompanying test(s). To complete a subsection or an entire section, a comprehensive test involving the appropriate functional modules is conducted to confirm a crucial substage of the project.

3 Audio Communication

The purpose of audio communication is to retrieve the modulated signal and digitize it for digital processing. After this digital processing, a binary message interpretable by the robot is obtained. The components used in this section are the dsPIC33FJ128MC802 and the UART, which is used to visualize the test results for this part using a Python code.

3.1 Audio Acquisition Chain

The ADC of the dsPIC33FJ128MC802 is responsible for analog-to-digital conversion in a conversion range from $0V$ to $3.3V$. Given this conversion range, it is imperative to install an amplification stage between the microphone and the ADC with an offset (a DC voltage). Indeed, the output voltage from the microphone used for reception is quite low and is centered around 0V. Below, we discuss the design and implementation of the analog setup, starting from the command frame to the guard filter of the setup.

3.1.1 Frame Command

Throughout this project, we employ a frequency modulation mode in which the frequency of the modulated signal varies between predetermined frequencies. This is known as Frequency Shift Keying (FSK) modulation. The modulated signal in the project fluctuates between two frequencies around a central value and can be formulated as follows:

$$\text{modulated signal } = \begin{cases} A_0 \sin\left(2\pi\left(f_p - \Delta_f\right)t\right) \Leftrightarrow \text{ modulating signal } = 0 \\ A_0 \sin\left(2\pi\left(f_p + \Delta_f\right)t\right) \Leftrightarrow \text{ modulating signal } = 1 \end{cases} \quad \text{where} \qquad (1)$$

$f_p = 1kHz$ the carrier frequency.
$\Delta f = 100Hz$ the frequency shifting.

At a period of $T_s = 100ms$, the modulating signal exhibits a bitstream consisting of a preamble, data, and a postamble. The data comprises a command (2 bits) accompanied by its parameter (8 bits). These latter two elements represent a 10-bit message that the robot's central unit (the dsPIC33FJ128MC802) must be able to decode using its analog setup followed by appropriate

digital processing.

3.1.2 Analog circuit

To capture the modulated signal, it is crucial to design and implement an audio acquisition chain capable of reproducing the shape of the received signal. To analyze this analog circuit, we propose discussing the first component of the circuit, which is the microphone.

The Microphone: Prescribed for use in this project is an *"Electret Condenser Microphone"* KECG2740PBJ. Several usage criteria are directly validated from its datasheet available in the UV, such as the *"Standard Operating Voltage"* ***2V***, the output impedance $R_L = \mathbf{2.2\ kHz}$, and the bias current $I_{pol} = \mathbf{0.5\ mA}$. Considering that the circuit is powered by the dsPIC (3.3V), which itself is supplied with 5V, it can be immediately concluded that the microphone operates within the specified operational range.

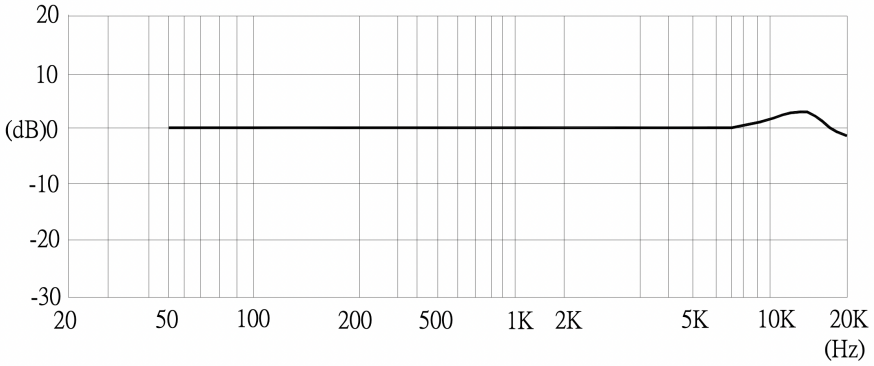

Figure 1: Frequency response curve of the KECG2740PB microphone

"Starting with the analysis of its frequency response (fig. 1), it can be observed that beyond 7kHz, the magnitude of the transfer function between the microphone output and the modulated signal exceeds 1. However, by limiting ourselves to project frequencies, i.e., $1.1kHz$ for the highest frequency, we conclude that the microphone gain is negligible since it is 0 dB. Similarly, assuming the microphone output to be at a maximum of 1mV in a quiet environment.

Secondly, we can proceed with the representation of the microphone circuit using its diagram in figure 2. Electret condenser microphones require an external power supply provided by the dsPIC. With its bias current not exceeding 0.5 mA, we can now calculate the resistance R1 using

the following equation: "

$$I_{pol} = \frac{V_{cc}}{2 \times (R1 + R2)} \quad \text{where} \tag{2}$$

$V_{cc} = 3.3V$.

$R2 = R_L = 2.2k\Omega$.

$I_{pol} = 0.5mA$.

Therefore, R1 must have a value greater than $1.1k\Omega$. This resistor, together with a capacitor C1, forms a low-pass filter circuit, filtering out any residues or other signals, allowing only the DC voltage to power the microphone.

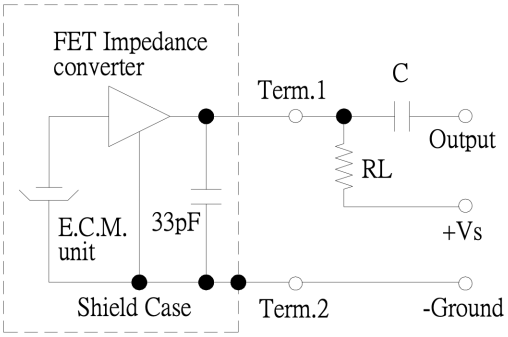

Figure 2: Microphone's electric circuit.

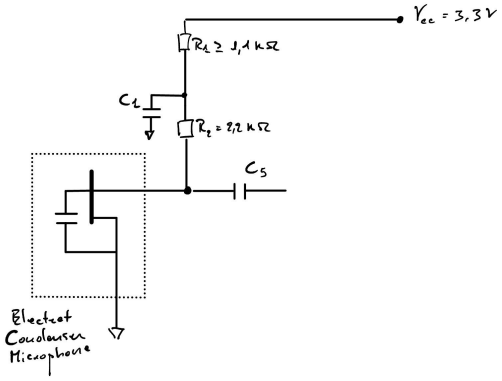

Figure 3: Low-pass filter with the mic.

For the capacitor design, we set a cutoff frequency $f_{Cutoff.1} = \frac{1}{2\pi \times R1 \times C1} = 90Hz$. As a reminder, the resistance R1 is greater than 1.1 kΩ. In lab UA5, we chose a resistance R1 of 1.5 kΩ, resulting in a capacitor value of C1 = 1.17 $\mu F \approx 1\mu F$. The capacitor in Figure **??** corresponds to capacitor C5 in Figure **??**, which filters the offset originating from the circuit and provides only the AC voltage at the microphone output to the rest of the circuit.

Amplification Stage: Amplification is provided by an operational amplifier (op-amp) supplied directly by lab UA5. It is the **MCP6271** op-amp powered with 3.3V and 0V at the output. Due to its power supply, an offset of 1.65V is introduced around which we can fully utilize the amplification range. This is achieved by a stage before the non-inverting input, composed of a resistive divider. Here it is, just below:

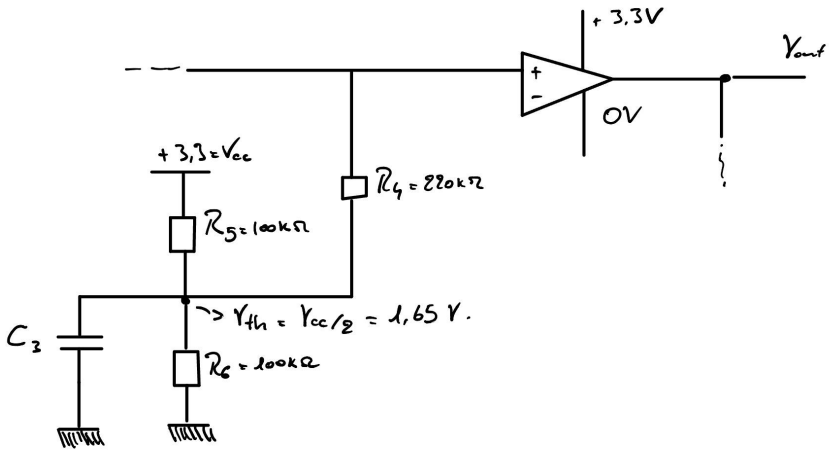

Figure 4: Voltage Divider

The values of R5 and R6 must be equal; we have chosen the ones proposed by the datasheet in the UV. The value of R4 is 220k Ω to obtain a sufficiently high input impedance at the input of the op-amp. This is to prevent the collapse of the signal from the microphone when working with alternating current. Meanwhile, C3 is chosen as a low-pass filter to eliminate high frequencies from the integrated circuit. It is determined in the same way as C1 with a cutoff frequency $f_{Cutoff.3} = 90Hz$. This gives us 17nF, which we approximate with a 15nF capacitor. C5 is then determined to act as a high-pass filter. Its cutoff frequency is set at 90Hz, approximately 7.9nF, which we approximate with a 10nF capacitor in lab UA5.

The amplifier gain is calculated assuming the superposition of the DC voltage gain with the AC voltage gain. It is given by the following formula:

$$G = (1 + \frac{R8}{R7}) \times (\frac{R4}{R4 + R2}) \quad \text{where} \tag{3}$$

$R8 \approx R4 + \frac{R5 \times R6}{R4 + R6}$

"Adhering to this, that is, a gain of 1650 given by $G = \frac{V_{out}}{V_{in}} = \frac{1.65V}{1mV}$, we can determine R7, which has a value of 200Ω. This calculated gain constitutes the sole amplification stage, as the gain of the guard filter stage is unity. Therefore, the op-amp in the first and only stage must provide all the necessary amplification.

Guard Filter: A guard filter is essential to prevent spectral folding. This is implemented using

an active Butterworth filter. It was designed using the 'Analog Filter Wizard' software and has the following form:"

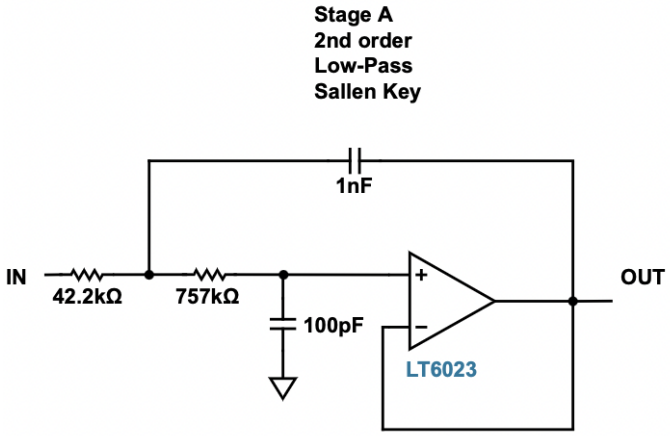

Figure 5: Electronic circuit of the guard filter

3.1.3 Validation of the audio acquisition chain

As visible in figures 11 and 12, the signals exiting the analog circuit have a peak-to-peak amplitude of 3.369V and an offset of approximately 1.68V, which is satisfactory considering the requirements outlined in the specifications.

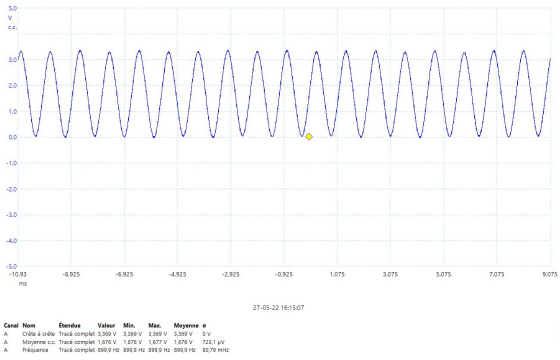

Figure 6: Analog signal at 900Hz

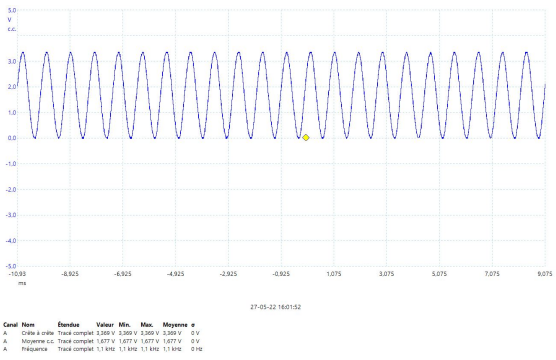

Figure 7: Analog signal at 1100 Hz

Here is a comprehensive representation of the electrical circuit, including the guard filter and all the resistors and capacitors calculated previously. The values of the components and capacitors

are summarized below.

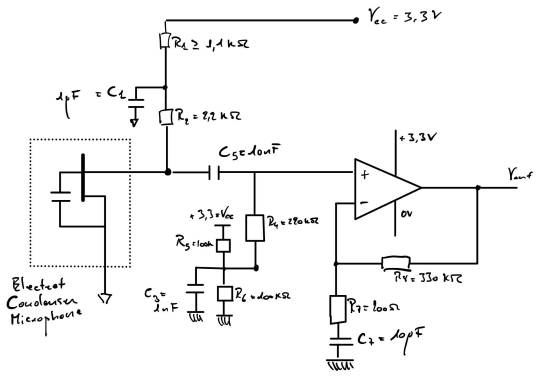

Figure 8: Analog circuit

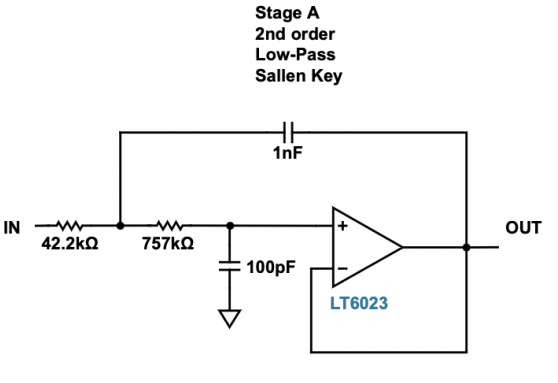

Figure 9: Guard filter

List of the electronic components		
Resistors/Capacitors	Theoretic values	Practical values
R1	$1.1k\Omega$	$1.5k\Omega$
R2	$2.2k\Omega$	$2.2k\Omega$
R4	$220k\Omega$	$220k\Omega$
R5	$100k\Omega$	$100k\Omega$
R6	$100k\Omega$	$100k\Omega$
R7	$200k\Omega$	$200k\Omega$
R8	$320k\Omega$	$320k\Omega$
C1	$1.17\mu F$	$1\mu F$
C3	$17nF$	$15nF$
C5	$7.9nF$	$10nF$
C7	$8.9\mu F$	$10\mu F$

3.2 Digital signal processing

3.2.1 Digital filters

The signal digitized by the ADC passes through 2 bandpass filters, one centered at a frequency of 900 Hz and the other at a frequency of 1100 Hz. If the signal does not contain the central frequency of the filter, it will be significantly attenuated. In reality, the two 8th-order

filters were each subdivided into 4 2nd-order filters. The filters were implemented using the following recursive equation:

$$\begin{cases} y_1(n) = \sum_{i=0}^{p} b_i.x(n-i) - \sum_{i=0}^{m} a_i.y_1(n-i) \\ y(n) = G.y_1(n) \end{cases}$$

The coefficients characterizing the filter were calculated using the provided Python code. This allowed the establishment of the 2 functional filters, and their output can be observed in Figure 14. Fixed-point arithmetic had to be implemented as well to achieve an execution time below the ADC sampling period.

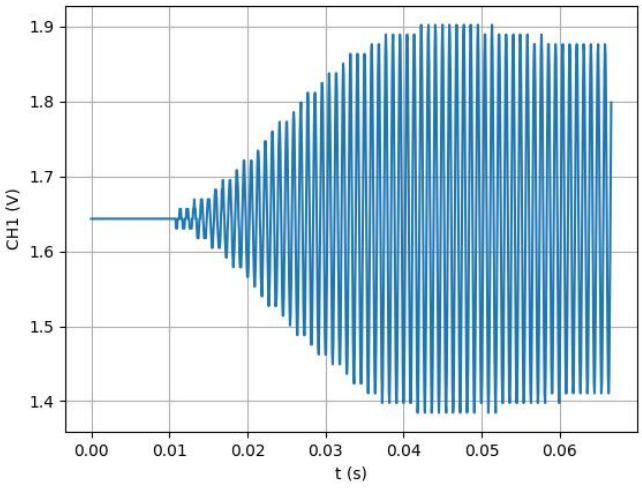

Figure 10: Filter's output at 1100 Hz

The validation of the digital filters was conducted through a Fourier transformation to display their frequency spectrum. As evident in the following figures, the signal at 1200 Hz is heavily attenuated, while the signal at 1100 Hz is not.

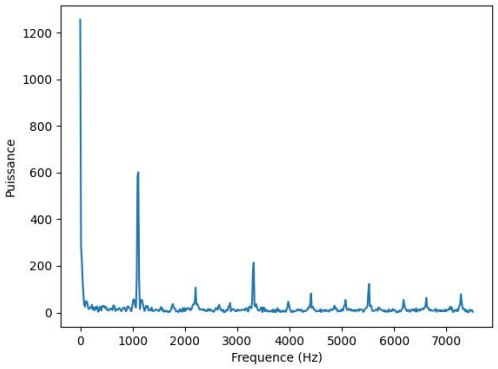

Figure 11: Filter's output at 1100 Hz. Frequency domain

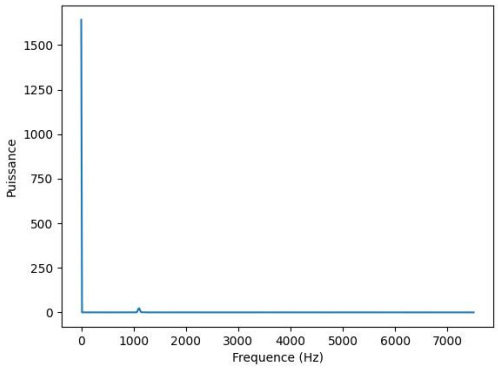

Figure 12: Filter's output at 1200 Hz. Frequency domain

3.2.2 The comparator and the FSK detector

The next step following the digital filters is the comparator. This component determines whether the received signal exhibits a frequency of f_0) or f_1) corresponding to a bit 0 or a bit 1, respectively. To achieve this, a comparison threshold is established through trial and error, typically set around 0.4-0.5V. The main reason for this choice is that the amplitudes of the signals attenuated by our digital filters generally do not exceed these values. In practice, the principle of this comparator is to compare the amplitude of samples to determine if it surpasses the imposed threshold. A certain number of bits are required to perform this amplitude comparison, which is determined based on the sampling period of 15,000Hz. To cover the entirety of one period of n samples, the following formula is established:

$$n \cdot T_s > T_{900Hz} \Leftrightarrow n \times 6,6667 \cdot 10^{-5} > 1,1111 \cdot 10^{-3} \Leftrightarrow n > 16,66 \tag{4}$$

$$n \cdot T_s > T_{1100Hz} \Leftrightarrow n \times 6,6667 \cdot 10^{-5} > 9,09 \cdot 10^{-4} \Leftrightarrow n > 13,64 \tag{5}$$

The number $n = 15$ is chosen. Finally, a frequency shift keying (FSK) sequence detector, referred to as the *fskDetector*, is invoked. This component decodes the Start Bit, the data frame, and, finally, the Stop Bit. At its output, the function returns the command (2 bits) along with the parameter (10 bits).

3.2.3 Exchange of information between the *dsPIC* via *UART*

The UART is an asynchronous communication protocol enabling two devices to communicate. All that is required is connecting the TX (transmission) pin of one device to the RX (reception) pin of another device, and vice versa.

This can be particularly useful between the dsPIC used for communication and a computer, allowing the validation of filter outputs, for instance. It is in this context that the Python code 'oscilloscope' was employed. Regarding communication between the communication dsPIC and the one responsible for vehicle movement, a similar approach is adopted, transmitting information from the communication TX pin to the movement RX pin. This is achieved using the following code:

```
message = fskDetector(detLow, detHigh);
if (message != 0){
    int16_t message2bits = (message & 0xFF00)>>8;
    int16_t message8bits = message & 0x00FF;
    printf("Message");
    while (U1STAbits.UTXBF){} // On attend que ça soit libre.
    U1TXREG = message2bits;
    while (U1STAbits.UTXBF){} // On attend à nouveau que ça soit libre.
    U1TXREG = message8bits;
}
```

Figure 13: Sending of the TX's information

```
while (1){ // On va vérifier en premier si l'UART1 a reçu un octet avec lequel on peut travailler. Code reprit de l'OSCILLOSCOPE.
    if(U1STAbits.URXDA){
        type = U1RXREG;
        //printf("type");
        if ((0 <= type) && (type < 4))
        {
            while (!U1STAbits.URXDA) {} // On attend que l'UART soit à nouveau disponible.
            parametre = U1RXREG;
            occupied = 1;
        }
    }
```

Figure 14: Reception of the TX's information

3.2.4 Validation of the acquisition audio channel

The validation process was recorded for ease of measurement, and the video is available on the GitHub link along with the codes and their comments

4 Vehicle movement

4.1 Description of the main components

The components required for the robot's movement are as follows:

- 2 direct current motors. These are equipped with a speed reducer and quadrature encoders. Powered by 5V, they provide two logic signals compatible with CMOS logic.
- 1 DRI004 driver. This driver serves as an interface between the dsPIC and the motors. Additionally, it has DIR1 and DIR2 pins determining the direction of the two wheels, along with PWM1 and PWM2 pins regulating their speed.
- 1 dsPIC33FJ128MC802. This microcontroller features two Output Compare modules for speed control and two Quadrature Encoder Interfaces used to retrieve the encoder registers required for closed-loop regulation.
- 1 battery NiMh of 7,2V.

4.2 Polar regulation

Polar regulation involves a combination of translational and rotational regulation. It is necessary to prevent the robot from following an arced path during simple translational regulation, for example. During translation, polar regulation ensures that the angular error remains zero. The same rationale applies to rotation. The schematic of polar regulation is depicted in Figure 15.

The error, i.e., the difference between the setpoint and the position measured by the encoders, enters a proportional controller with gains K_{pr} for rotation and K_{pl} for translation. The output of the controller is expressed as a duty cycle d_{cr} for rotation and d_{cl} for translation. Consequently, their sum constitutes the control signal for the left motor, while their difference regulates the right motor, achieved through the PWM modules. The encoders provide the robot's position, allowing for error calculation. Finally, feedback adjusts the speed to reach the setpoint.

Accuracy and Dead Zone

Due to its nature, this regulation will never practically achieve the setpoint precisely. The speed

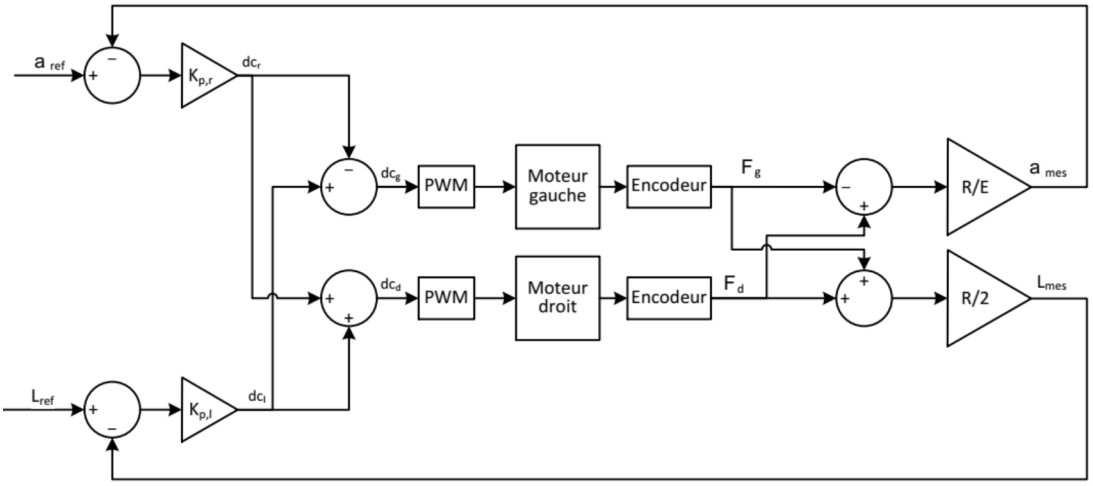

Figure 15: Scheme of the polar regulation

profile includes deceleration once the robot is close to its destination. Thus, the closer it gets, the slower it moves; the duty cycle decreases. This reduction results in a very low voltage applied to the wheels, which is no longer sufficient to overcome wheel inertia, resistive torque, friction, etc. The robot stops moving. The difference between its final position and the setpoint is commonly known as the dead zone. It is essential to evaluate it and consider strategies to minimize it, especially when evaluating the robot's performance.

4.2.1 Set-point Generation

The imposed velocity profile follows the shape described in the figure. 16.

Il s'agit d'un profil trapézoïdal composé d'une phase d'accélération , une phase de vitesse constante et une décélération.

Méthode 1

The first method involves trajectory tracking. It requires integrating the position profile. Consequently, at regular intervals (hence the need for a timer in interrupt mode), the setpoint is generated based on the elapsed time. This approach has the advantage of gradually increasing the duty cycle because the difference between the robot's position at time t and the setpoint at the same time t is relatively small. However, it demands more resources from the dsPIC.

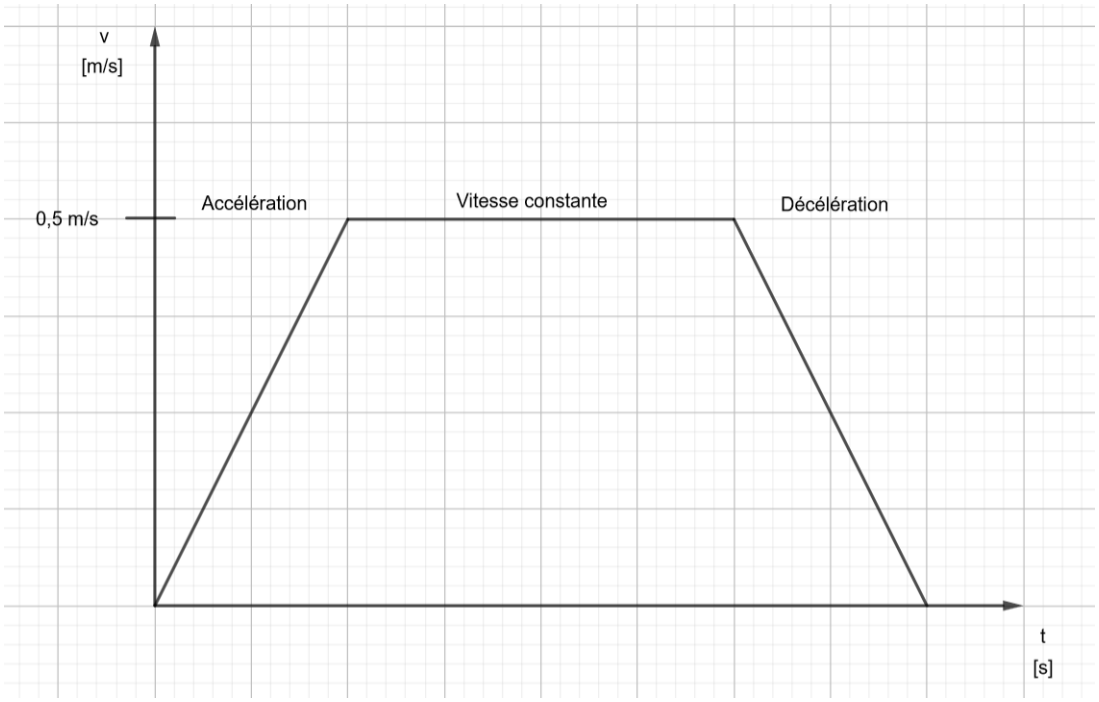

Figure 16: Profile of the robot's speed

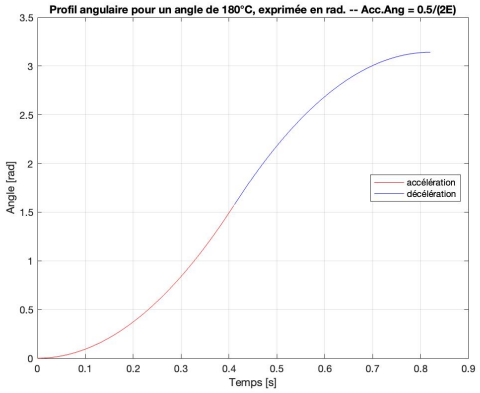

Figure 17: Angular profile

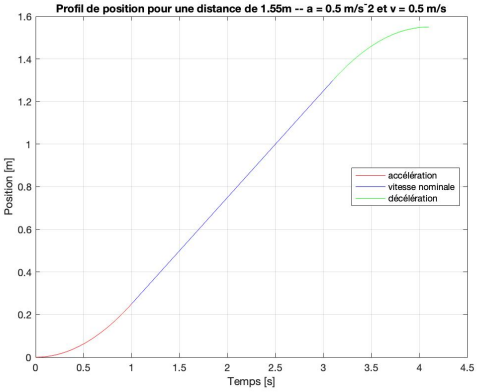

Figure 18: Position profile

Method 2

To simulate this profile, another option is to define the duty cycle as proportional to a position error with the reference now fixed to its final value. However, this means that the duty cycle will be very high from the beginning because the position error will be maximum initially. To

overcome this issue, it is appropriate to limit the translation and rotation duty cycles to 0.5. This especially allows the possibility of controlling the angle during translation because the total duty cycle (the sum of both) will not be saturated at 1, enabling full utilization of polar regulation.

4.2.2 Proportional Controller

As explained earlier, the implemented controller is a proportional controller with gain K_{pr} for rotation and K_{pl} for translation. The gains were obtained through phase and gain margin, set at 30 degrees and 6 dB, respectively. Therefore, the provided gain K_{pl} is 3.69 m^{-1}. This gain needs to be converted to radians for K_{pr}.

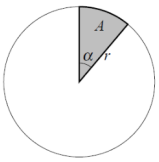

Figure 19: Arc of circle

Knowing that the radius of the arc formed by the rotation of the robot is half of the wheelbase E, the relationship between the angle and the arc of the circle formed is as follows:

$$\alpha = \frac{2}{E} * A \tag{6}$$

So the relationship between K_{pr} and K_{pl} is:

$$K_{pr} = \frac{E}{2} * K_{pl} \tag{7}$$

4.2.3 Encoders and Feedback

Conversion of numeric values provided by the registers

The data provided by the encoders are stored in two registers, POS1CNT and POS2CNT, which increment as the robot moves. Given in hexadecimal, it is necessary to convert the values

stored in these registers to obtain measurements in appropriate units. First, a cast operation is used to convert these values into integers. Since these are quadrature encoders, the values in the registers need to be divided by 4. The obtained measurement is then in degrees and corresponds to the set values F_g and F_d in Figure 15.

So, the 'measurements' provided for feedback are expressed as follows:

$$L_{mes} = \frac{R}{2}\frac{\pi}{180}(F_d + F_g) \tag{8}$$

$$\alpha_{mes} = \frac{R}{E}\frac{\pi}{180}(F_d - F_g) \tag{9}$$

4.3 Validation of the Movement Section

The regulation using 'Method 2' has been successful, yielding satisfactory results. However, the robot tends to deviate slightly in an arc during translation despite the polar regulation. Several hypotheses could explain this, such as a rotation-related gain being too low or saturation due to the translation setpoint.

4.3.1 Dead Zone Handling and Precision

In order to overcome the dead zone issues, an offset was introduced to the initial set-points. This offset was determined iteratively and empirically for both translation and rotation. It involves multiplying the initial setpoint by a certain factor to achieve a precision on the order of centimeters for translation and degrees for rotation. This approach significantly improved the robot's accuracy and minimized deviations, ensuring more precise movement and control.

5 Conclusion

In conclusion, the development of this robot has facilitated the acquisition of a plethora of essential knowledge in the field of electronic engineering. From analog electronics to control systems, encompassing telecommunications and instrumentation, this project has underscored the significance of all the disciplines learned throughout this engineering education.

6 Appendix

6.1 GitHub Link

https://github.com/Alex1721/integrated-project.git

6.2 Robot movement (second variant)

```
/*
 * File:   main.c
 * Author: User
 *
 * Created on 4 mai 2022, 22:23
 */

//Right  wheel = 1
//Left wheel = 2
#include "main.h"
#include "math.h"

int main(){
    frcPll40MHzConfig();
    int16_t Tram = 0b1100001111;
    //Instruction
    int16_t Intruction = 0b00;
    float Dist = 100;  //[cm]
    float Angle = 0; //[ ]
    //You'll need to make a condition on the dist/angle to be sure to have [m] or
    [  ]
    float error[2] ={Dist,Angle};
    //setups
    setup();

while(1){
    POS1CNT = 0x0000;
    POS2CNT = 0x0000;
    //Get the register of the wheel (1 and 2)

    if(Intruction == 0b00){
        _LATB12 = 1; //Dir1 (forward with 3.3)
```

```
32          _LATB11 = 0; //Dir2 (forward with 0)
33
34          while(error[0] > 0.5f){
35
36              polar(error[0], error[1]);
37              error[0] = errorLength(Dist);
38              error[1] = errorAngular(Angle);
39              if(error[0]<1.0f)
40              {
41                  error[0]=0;
42                  error[1]=0;
43                  polar(error[0],error[1]);
44                      LATBbits.LATB4  = 1;
45              }
46          }
47      }
48          if(Intruction == 0b01){
49          _LATB12 = 0; //Dir1 (forward with 3.3)
50          _LATB11 = 1; //Dir2 (forward with 0)
51
52          while(error[0] < -0.5f){
53              polar(error[0], error[1]);
54              error[0] = errorLength(Dist);
55              error[1] = errorAngular(Angle);
56              if(error[0] > -1.5f){
57                  error[0]=0;
58                  error[1]=0;
59                  polar(error[0],error[1]);
60
61              }
62          }
63      }
64              if(Intruction == 0b11){
65          _LATB12 = 1; //Dir1 (forward with 3.3)
66          _LATB11 = 1; //Dir2 (forward with 0)
67
68          while(error[1] > 0.5f){
69              polar(error[0], error[1]);
70              error[0] = errorLength(Dist);
71              error[1] = errorAngular(Angle);
72
```

```
            if(error[1] < 1.0f)
            {
                error[0]=0;
                error[1]=0;
                polar(error[0],error[1]);

            }


        }
    }

    if(Intruction == 0b10){
        _LATB12 = 0; //Dir1 (forward with 3.3)
        _LATB11 = 0; //Dir2 (forward with 0)

        while(error[1] < -0.5f){
            polar(error[0], error[1]);
            error[0] = errorLength(Dist);
            error[1] = errorAngular(Angle);

            if(error[1] > -5.0f){
                error[0]=0;
                error[1]=0;
                polar(error[0],error[1]);
            }


        }
    }
}
    return 0;
}

void polar(float errorLength, float errorAngle){


    float dcr = errorAngle * Kpr;
    float dcl = errorLength * Kpl;

        if (dcl > 0.5f){
```

```
114         dcl = 0.5f;
115         }
116         else if (dcl < -0.5f){
117             dcl = -0.5f;
118 
119         }
120         if (dcr > 0.5){
121         dcr = 0.5;
122         }
123         else if (dcl < -0.5f){
124             dcr = -0.5f;
125 
126         }
127 
128 
129 
130     float dcd = dcl + dcr;
131     float dcg = dcl - dcr;
132 
133     if(dcd < 0.0f){
134         _LATB12= 0;
135         OC1RS = -(dcd)*PR2;
136     }
137     else{
138         OC1RS = (dcd)*PR2;
139 
140     }
141     if(dcg < 0.0f){
142         _LATB11= 1;
143         OC2RS = -(dcg)*PR2;
144     }
145     else{
146         OC2RS = (dcg)*PR2;
147     }
148 }
149 
150 float errorAngular(float angleRef){
151     angleRef = 1.5*angleRef;
152     float encod1 = ((int)POS1CNT)/(4);
153     float encod2 = ((int)POS2CNT)/(4);
154 
```

```
155     float angleMes = ((-encod1+encod2) *R) / E ;
156     float errorAngle = angleRef + angleMes;
157     return errorAngle;
158 }
159
160 float errorLength(float lengthRef){
161     float encod1 = ((int)POS1CNT*PI*R)/(4*180);
162     float encod2 = ((int)POS2CNT*PI*R)/(4*180);
163
164     float lengthMes = ((encod2+encod1) ) / 2.0f ;
165     float errorLength = lengthRef - lengthMes;
166     return errorLength;
167 }
168
169 //      ////////////////////////////////////////////////////////////////////////////////////////////////////////////
170
171 float errorwheelB(float dist, float errorL[]){
172
173     //return the error of 1
174
175
176         float encod1 = (int)POS1CNT*PI*R/(4*180);
177         float encod2 = (int)POS2CNT*PI*R/(180*4);
178         float moyencod = (encod1 + encod2) / 2.0f;
179         float error = dist - moyencod;
180
181
182         if (errorL[1] < 2.0f ){//
183             error = 0;
184         }
185
186
187
188
189           return error;
190
191 }
192
193
```

```
194
195
196 void setup(){
197
198     AD1PCFGL = 0xFFFF; //set all the pins on 1
199
200
201     //Set the pins of the encoder
202
203     //QEI 1
204     QEI1CONbits.QEIM = 0b111; //Fait registre pour data
205     RPINR14bits.QEA1R = 3;
206     RPINR14bits.QEB1R = 2;
207
208     //QEI2
209     QEI2CONbits.QEIM = 0b111;
210     RPINR16bits.QEA2R = 14;
211     RPINR16bits.QEB2R = 13;
212
213     //Pin Dir
214     TRISBbits.TRISB12 = 0;
215     TRISBbits.TRISB11 = 0;
216     TRISBbits.TRISB4 = 0;
217     TRISBbits.TRISB3 = 0;
218     _LATB12 = 1; //Dir1 (forward with 3.3)
219     _LATB11 = 1; //Dir2 (forward with 0)
220     _LATB4 = 0;
221     _LATB3 = 0;
222
223     // Initialize and enable Timer2
224     PR2 = 7999; // Load the period value; 40MHz*200us -1 = PR2 ->7999
225
226     //PWM1
227
228     //Driver setup
229     OC1RS = 0; // Write the duty cycle for the second PWM pulse
230     OC1R = 0;
231     OC1CONbits.OCM = 0b110; // Select the Output Compare mode
232     OC1CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
233         // Initialize Output Compare Module
```

```
234     _RP9R = 0b10010; //configure the pin to setup the pwm1 (for the second pwm,
        set the LSB at 1)
235     T2CONbits.TON = 1; // Start Timer
236
237
238     //PWM2
239
240     //Driver setup
241     OC2RS = 0; // Write the duty cycle for the second PWM pulse
242     OC2R = 0;
243     OC2CONbits.OCM = 0b110; // Select the Output Compare mode
244     OC2CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
245         // Initialize Output Compare Module
246     _RP10R = 0b10010; //configure the pin to setup the pwm1 (for the second pwm,
        set the LSB at 1)
247     T2CONbits.TON = 1; // Start Timer
248
249
250 }
251
252 // DSPIC33FJ128MC802 Configuration Bit Settings
253 // 'C' source line config statements
254 // FBS
255 #pragma config BWRP = WRPROTECT_OFF     // Boot Segment Write Protect (Boot
        Segment may be written)
256 #pragma config BSS = NO_FLASH           // Boot Segment Program Flash Code
        Protection (No Boot program Flash segment)
257 #pragma config RBS = NO_RAM             // Boot Segment RAM Protection (No Boot
        RAM)
258
259 // FSS
260 #pragma config SWRP = WRPROTECT_OFF     // Secure Segment Program Write Protect (
        Secure segment may be written)
261 #pragma config SSS = NO_FLASH           // Secure Segment Program Flash Code
        Protection (No Secure Segment)
262 #pragma config RSS = NO_RAM             // Secure Segment Data RAM Protection (No
        Secure RAM)
263
264 // FGS
265 #pragma config GWRP = OFF               // General Code Segment Write Protect (
        User program memory is not write-protected)
```

```
266 #pragma config GSS = OFF                // General Segment Code Protection (User
        program memory is not code-protected)
267
268 // FOSCSEL
269 #pragma config FNOSC = FRC              // Oscillator Mode (Internal Fast RC (FRC)
        )
270 #pragma config IESO = OFF               // Internal External Switch Over Mode (
        Start-up device with user-selected oscillator source)
271
272 // FOSC
273 #pragma config POSCMD = NONE            // Primary Oscillator Source (Primary
        Oscillator Disabled)
274 #pragma config OSCIOFNC = OFF           // OSC2 Pin Function (OSC2 pin has clock
        out function)
275 #pragma config IOL1WAY = ON             // Peripheral Pin Select Configuration (
        Allow Only One Re-configuration)
276 #pragma config FCKSM = CSECMD           // Clock Switching and Monitor (Both Clock
         Switching and Fail-Safe Clock Monitor are disabled)
277
278 // FWDT
279 #pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler (1:32,768)
280 #pragma config WDTPRE = PR128           // WDT Prescaler (1:128)
281 #pragma config WINDIS = OFF             // Watchdog Timer Window (Watchdog Timer
        in Non-Window mode)
282 #pragma config FWDTEN = OFF             // Watchdog Timer Enable (Watchdog timer
        enabled/disabled by user software)
283
284 // FPOR
285 #pragma config FPWRT = PWR128           // POR Timer Value (128ms)
286 #pragma config ALTI2C = OFF             // Alternate I2C  pins (I2C mapped to SDA1
        /SCL1 pins)
287 #pragma config LPOL = ON                // Motor Control PWM Low Side Polarity bit
         (PWM module low side output pins have active-high output polarity)
288 #pragma config HPOL = ON                // Motor Control PWM High Side Polarity
        bit (PWM module high side output pins have active-high output polarity)
289 #pragma config PWMPIN = ON              // Motor Control PWM Module Pin Mode bit (
        PWM module pins controlled by PORT register at device Reset)
290
291 // FICD
292 #pragma config ICS = PGD1               // Comm Channel Select (Communicate on
        PGC1/EMUC1 and PGD1/EMUD1)
```

```
293 #pragma config JTAGEN = OFF              // JTAG Port Enable (JTAG is Disabled)
294
295 // #pragma config statements should precede project file includes.
296 // Use project enums instead of #define for ON and OFF.
297
298 #include <xc.h>
299
300
301 /* Configures PLL prescaler, PLL postscaler, PLL divisor to obtain Fosc = 80MHz
302  * with the FRC oscillator (Fin = 7.37MHz).
303  * We obtain Fosc = 7.37MHz*65/(*2) = 79.96MHz */
304 void frcPll40MHzConfig(void) {
305   // Fosc = Fin*M/(N1*N2), where :
306   //     M = PLLFBD + 2
307   //     N1 = PLLPRE + 2
308   //     N2 = 2 x (PLLPOST + 1)
309     PLLFBD = 63;
310     CLKDIVbits.PLLPRE = 1;
311     CLKDIVbits.PLLPOST = 0;
312
313   // Initiate Clock Switch to FRC with PLL
314   __builtin_write_OSCCONH( 0x01 );
315   __builtin_write_OSCCONL( OSCCON | 0x01 );
316   // Wait for Clock switch to occur
317     while (OSCCONbits.COSC != 0b001);
318     // Wait for PLL to lock
319     while(OSCCONbits.LOCK != 1);
320 }
```

Listing 1: Code exemple pour le déplacement du robot