Hardware Challenge

Author: Meseșan Bogdan Cristian

Components:

-10:1 Micro Metal Gearmotor HPCB 6V with Extended Motor Shaft

-Magnetic Encoder, compatible with HPCB motors, with 12 pulses/revolution

-L293D motor driver

-ATMEGA328P microcontroller

-LM044L LCD

-3 resistors of 10k ohm

-3 buttons

-Pinion with 60 teeth and 30mm diameter

-Rack with 62 teeth and 101 mm length

Software:

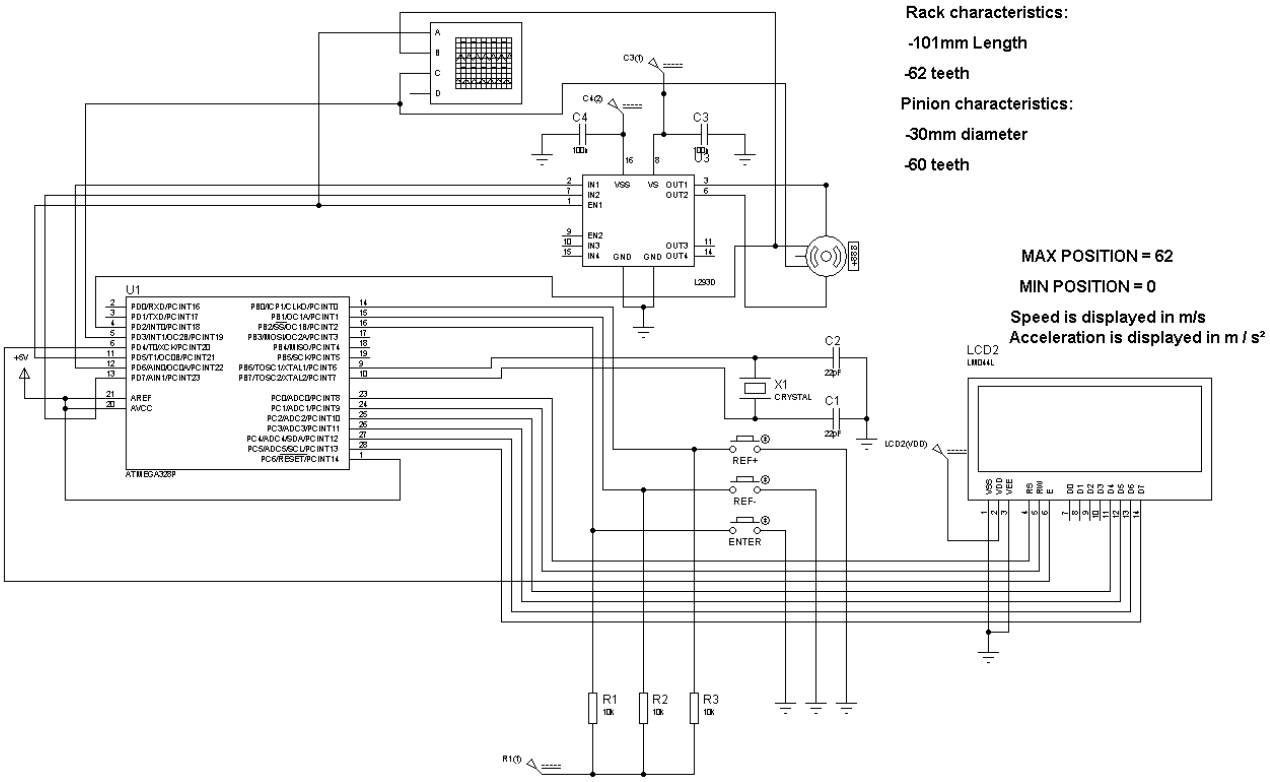
-The program is written in C and was developed using Atmel Studio. The source code has three modules: LCD module, Motor module and Main program.

-The entire system is simulated using Proteus.

-Link to the project which includes a video demonstration:

https://drive.google.com/open?id=1IcRmnjPMFKoikTBz1TI52WN09gHtmCMA

Schematic:


Rack characteristics:
-101mm Length
-62 teeth
Pinion characteristics:
-30mm diameter
-60 teeth
MAX POSITION = 62
MIN POSITION = 0
Speed is displayed in m/s
Acceleration is displayed in m / s²
U1
ATMEGA328P
U3
L293D
LCD2
LM044L
X1
CRYSTAL
C1
C2
C3
C4
R1
R2
R3
REF+
REF-
ENTER
+5V

Source code:

1. LCD module

```
/*
 * LCD.h
 *
 * Created: 3/24/2020 2:12:23 PM
 *  Author: Bogdan
 */


#ifndef LCD_H_
#define LCD_H_

#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>

//Sends the Enable signal to the LCD
void LCD_EN(void);

//Sends a 4-bit instruction to the LCD
void LCD_SendInstr(unsigned char val);

//Sends an 8-bit data value, which will be displayed on the LCD
void LCD_WriteData(unsigned char val);

//Sends all the necessary setup instructions to the LCD
void LCD_Setup(void);

//Writes a signed integer number on the LCD
void LCD_WriteInt(int num);

//Writes a string on the LCD
void LCD_WriteString(char *str);

//Writes a rational number, with 4 digit accuracy on the LCD
void LCD_WriteDouble(double num);

//Sets the position of the next character written to the LCD
void LCD_SetPosition(unsigned char pos);

//returns the mirrored number
unsigned int mirr_number(unsigned int num);

//returns the number of digits in a number
unsigned int nr_cif(unsigned int num);

//Writes the initial display
//A = Acceleration
//V = Velocity or Speed
//Pos = the current position of the pinion on the rack
```

```
//Ref = the desired position
void LCD_InitialDisplay(void);
void LCD_DisplayAcc(double acc);
void LCD_DisplayVel(double vel);
void LCD_DisplayPos(int pos);
void LCD_DisplayRef(int ref);

#endif /* LCD_H_ */
```

```
/*
 * LCD.c
 *
 * Created: 3/24/2020 2:13:04 PM
 *  Author: Bogdan
 */

 #include "LCD.h"

 //Sends the Enable signal to the LCD
 void LCD_EN(void)
 {
        PORTD &= ~(1 << PORTD4);
        PORTD |= (1 << PORTD4);
        _delay_us(40);
        PORTD &= ~(1 << PORTD4);
 }

//Sends a 4-bit instruction to the LCD
 void LCD_SendInstr(unsigned char val)
 {
        //Set RS and R/W on 0
        PORTC &= ~(1 << PORTC0 | 1 << PORTC1);
        PORTC &= ~(0x0F << 2);
        PORTC |= (val << 2);
        LCD_EN();
 }

//Sends an 8-bit data value, which will be displayed on the LCD
 void LCD_WriteData(unsigned char val)
 {
        unsigned char aux;

        PORTC &= ~(1 << PORTC0 | 1 << PORTC1);
        PORTC &= ~(0x0F << 2);

        //Set RS to 1 and R/W on 0
        PORTC |= (1 << PORTC0);

        //send the H part = the most significant 4 bits
        aux = (val >> 4);
        PORTC &= ~(0x0F << 2);
        PORTC |= (aux << 2);
        LCD_EN();
```

```
        //send the L part = the least significant 4 bits
        aux = (val << 4);
        aux = (aux >> 4);
        PORTC &= ~(0x0F << 2);
        PORTC |= (aux << 2);
        LCD_EN();
 }

//Sends all the necessary setup instructions to the LCD
 void LCD_Setup(void)
 {
        unsigned char instr[8] = {3, 3, 3, 2, 2, 1, 0, 12};
        unsigned char i;

        for (i = 0; i < 8; i++)
        {
               LCD_SendInstr(instr[i]);
               _delay_ms(1);
        }
 }

//Writes a signed integer number on the LCD
 void LCD_WriteInt(int num)
 {
        unsigned int val;
        unsigned int cif;

        if (num < 0)
        {
               LCD_WriteData('-');
               num = num * (-1);
        }

        //we send the digits in reverse order
        val = mirr_number(num);
        //in case the number ends with a 0 we need to know the number of digits it has
        cif = nr_cif(num);

        while (val > 0)
        {
               //write digit by digit
               LCD_WriteData(val % 10 + 48);
               val = val / 10;
               cif--;
        }
        if (cif > 0 || num == 0)
        {
               //case in which the original number ends with a '0' digit, we send out one
more '0'
               LCD_WriteData('0');
               if (cif > 0)
                     cif--;
               while (cif > 0)
               {
                     LCD_WriteData('0');
                     cif--;
               }
        }
```

```
}

//returns the mirrored number of the input
unsigned int mirr_number(unsigned int num)
{
	unsigned int val = 0;
	while (num > 0)
	{
		val = val * 10 + num % 10;
		num = num / 10;
	}
	return (val);
}

//returns the number of digits in a given number
unsigned int nr_cif(unsigned int num)
{
	unsigned int sum = 0;

	if (num == 0)
		return (1);

	while (num > 0)
	{
		sum++;
		num = num / 10;
	}

	return (sum);
}

//Writes a string on the LCD
void LCD_WriteString(char *str)
{
	unsigned int i = 0;

	while (str[i] != 0)
	{
		LCD_WriteData(str[i]);
		i++;
	}
}

//Writes a rational number, with 4 digit accuracy on the LCD
void LCD_WriteDouble(double num)
{
	int aux;

	if (num < 0)
	{
		LCD_WriteData('-');
		num = num * (-1);
	}
	//write the integer part of the number
	aux = (int)(num);
	LCD_WriteInt(aux);

	//find the rational part of the number and display it
```

```
        num = (double)(num - aux);
        num = num * 10000;
        aux = (int)(num);
        LCD_WriteData('.');
        if (aux < 1000)
        {
              LCD_WriteInt(0);
              if (aux < 100)
              {
                    LCD_WriteInt(0);
                    if (aux < 10)
                    {
                          LCD_WriteInt(0);
                          LCD_WriteInt(aux);
                    }
                    else
                    {
                          LCD_WriteInt(aux);
                    }
              }
              else
              {
                    LCD_WriteInt(aux);
              }
       }
       else
       {
              LCD_WriteInt(aux);
       }
 }

//Sets the position of the next character written to the LCD
 void LCD_SetPosition(unsigned char pos)
 {
        unsigned char aux;

        //Send High value
        aux = 0b00001000 + (pos >> 4);
        LCD_SendInstr(aux);

        //Send Low value
        aux = (pos << 4);
        aux = (aux >> 4);
        LCD_SendInstr(aux);
 }

//Writes the initial display
//A = Acceleration
//V = Velocity or Speed
//Pos = the current position of the pinion on the rack
//Ref = the desired position
 void LCD_InitialDisplay(void)
 {
       LCD_SetPosition(0);
       LCD_WriteString("V:");

       LCD_SetPosition(12);
       LCD_WriteString("Pos:");
```

```
	LCD_SetPosition(20);
	LCD_WriteString("A:");

	LCD_SetPosition(32);
	LCD_WriteString("Ref:");
}

 void LCD_DisplayAcc(double acc)
 {
	unsigned int pos = nr_cif((int)(acc)) + 22 + 5;

	LCD_SetPosition(22);
	LCD_WriteDouble(acc);

	if (acc < 0)
		pos++;
	LCD_SetPosition(pos);
	while (pos < 32)
	{
		LCD_WriteData(' ');
		pos++;
	}
 }

 void LCD_DisplayVel(double vel)
 {
	unsigned int pos = nr_cif((int)(vel)) + 2 + 5;

	LCD_SetPosition(2);
	LCD_WriteDouble(vel);

	LCD_SetPosition(pos);
	while (pos < 12)
	{
		LCD_WriteData(' ');
		pos++;
	}
 }

void LCD_DisplayPos(int pos)
{
	unsigned int pos1 = nr_cif((int)(pos)) + 16;

	LCD_SetPosition(16);
	LCD_WriteInt(pos);

	LCD_SetPosition(pos1);
	while (pos1 < 20)
	{
		LCD_WriteData(' ');
		pos1++;
	}
}

void LCD_DisplayRef(int ref)
{
	unsigned int pos = nr_cif((int)(ref)) + 36;
```

```
	LCD_SetPosition(36);
	LCD_WriteInt(ref);

	LCD_SetPosition(pos);
	while (pos < 40)
	{
		LCD_WriteData(' ');
		pos++;
	}
}
```

2. Motor module

```
/*
 * Motor.h
 *
 * Created: 3/24/2020 3:07:53 PM
 *  Author: Bogdan
 */


#ifndef MOTOR_H_
#define MOTOR_H_

#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "LCD.h"

//the PWM is between 0 and 255
#define MAX_PWM 255

//nr of pulses/rotation generated by the encoder
#define PULSES 12.0

//gear ratio of the motor
#define GEAR_RATIO 10.0

//constant used to calculated the velocity
#define VELOCITY_CONST (double)(((100000 / PULSES) / GEAR_RATIO) * PINION_RADIUS)

//the radius of the pinion in meters; in this case it is 15 mm
#define PINION_RADIUS 0.015

//number of teeth of the pinion
#define PINION_TEETH 60

//constant used to calculate the pinion position
#define PINION_CONSTANT (int)((PULSES * GEAR_RATIO) / PINION_TEETH)

//number of the rack teeth
#define RACK_TEETH 62
```

```
//Maximum/Minimum position of the encoder pulses
#define MAX_POSITION (RACK_TEETH * PINION_CONSTANT)
#define MIN_POSITION 0

//PID controller constants, found by tuning the controller
#define KP 0.5
#define KI 0.15
#define IMAX_LIM 4
#define KD 0.5

void setupPorts(void);

//Timer0 is used to generate the PWM signal which drives the motor
//The duty factor is between 0 and 255
void setupTimer0(void);
void startTimer0(void);
void stopTimer0(void);

//Timer1 is programmed to generate an interrupt at every 10 microseconds
//I use this value to calculate speed and acceleration
void setupTimer1(void);
void setupTimer1Int(void);
void startTimer1(void);
void stopTimer1(void);

//Int0 and Int1 are used for the signals coming from the encoder
//They are active on the Rising edge
void setupInt0(void);
void setupInt1(void);
//Pin Change Interrupt on PortB, used for the user interface buttons
void setupPinChgInt(void);


void MotorSetup(void);
void MotorSetDuty(unsigned char duty);
void MotorSetDirection(unsigned char dir);
void MotorStop(void);

//get the necessary direction based on the reference position and the actual position
unsigned int getDirection(int ref, int pos);
//Get the error = Set_Point - Process_Variable
int getError(int sp, int pv);
//Get the necessary PID drive
int getPIDdrive(int error, int prev_error, double *errorI);
#endif /* MOTOR_H_ */
```

```c
/*
 * Motor.c
 *
 * Created: 3/24/2020 3:08:56 PM
 *  Author: Bogdan
 */
 #include "Motor.h"

void setupPorts(void)
{
	//LCD ports
	//RS and R/W
	DDRC |= (1 << DDC0 | 1 << DDC1);
	//D7, D6, D5, D4
	DDRC |= (1 << DDC2 | 1 << DDC3 | 1 << DDC4 | 1 << DDC5);
	//EN
	DDRD |= (1 << DDD4);

	//Timer0 port
	DDRD |= (1 << DDD5);

	//Motor direction control
	DDRD |= (1 << DDD6 | 1 << DDD7);

	//Interrupts input ports
	DDRD &= ~(1 << DDD2);
	DDRD &= ~(1 << DDD3);

	//Buttons input pins
	DDRB = ~(1 << DDB0 | 1 << DDB1 | 1 << DDB2);
}


//Timer0 is used to generate the PWM signal which drives the motor
//The duty factor is between 0 and 255
 void setupTimer0(void)
 {
	//Set Timer0 in Fast PWM mode, inverting type
	 TCCR0A |= (1 << COM0B1 | 1 << COM0B0 | 1 << WGM01 | 1 << WGM00);
	 TCCR0B |= (1 << WGM02);
	 //OCR0A will set the Maximum PWM
	 OCR0A = MAX_PWM;
	 //OCR0B will set the duty cycle
	 OCR0B = MAX_PWM;
 }

 void startTimer0(void)
 {
	 //Prescaler N = 256
	 TCCR0B |= (1 << CS02 | 0 << CS01 | 0 << CS00);
 }

 void stopTimer0(void)
 {
	 TCCR0B &= !(1 << CS02 | 0 << CS01 | 0 << CS00);
 }
```

```
//Timer1 is programmed to generate an interrupt at every 10 microseconds
//I use this value to calculate speed and acceleration
void setupTimer1(void)
{
      //Set CTC mode with compare set on OCR1A
       TCCR1B |= (1 << WGM12);
       OCR1AH = 0X00;
       OCR1AL = 159;
}

//Activate Timer1 interrupt on compare with OCR1A
void setupTimer1Int(void)
{
       TIMSK1 |= (1 << OCIE1A);
}

void startTimer1(void)
{
      //Prescaler N = 1
       TCCR1B |= (0 << CS12 | 0 << CS11 | 1 << CS10);
}

void stopTimer1(void)
{
       TCCR1B &= ~(1 << CS12 | 1 << CS11 | 1 << CS10);
}

//Int0 and Int1 are used for the signals coming from the encoder
//They are active on the Rising edge
void setupInt0(void)
{
       DDRD &= ~(1 << DDB2);
       EICRA |= (1 << ISC01 | 1 << ISC00);
       EIMSK |= (1 << INT0);
}

void setupInt1(void)
{
       DDRD &= ~(1 << DDB3);
       EICRA |= (1 << ISC11 | 1 << ISC10);
       EIMSK |= (1 << INT1);
}

//Pin Change Interrupt on PortB, used for the user interface buttons
void setupPinChgInt(void)
{
      PCMSK0 |= (1 << PCINT0 | 1 << PCINT1 | 1 << PCINT2);
      PCICR |= (1 << PCIE0);
}


void MotorSetup(void)
{
       MotorSetDirection(0);
       setupTimer1();
       setupTimer1Int();
       setupTimer0();
       setupInt0();
```

```
        setupInt1();
        setupPinChgInt();
        sei();
}

void MotorSetDuty(unsigned char duty)
{
      //This operation is needed because the Timer is set inverting mode
       OCR0B = MAX_PWM - duty;
}

void MotorSetDirection(unsigned char dir)
{
        if (dir == 1)
        {
                PORTD &= ~(1 << PORTD6 | 1 << PORTD7);
                PORTD |= (1 << PORTD6);
        }
        else
        {
                if (dir == 0)
                {
                        PORTD &= ~(1 << PORTD6 | 1 << PORTD7);
                        PORTD |= (1 << PORTB7);
                }
        }
}

void MotorStop(void)
{
      MotorSetDuty(0);
}

//get the necessary direction based on the reference position and the actual position
unsigned int getDirection(int ref, int pos)
{
      if (ref > pos)
              return (1);
      else
              return (0);
}


//Get the error = Set_Point - Process_Variable
int getError(int sp, int pv)
{
      int aux = sp * PINION_CONSTANT;

      if (sp > RACK_TEETH)
              aux = MAX_POSITION;
      if (sp < 0)
              aux = 0;

      return (aux - pv);
}

//Get the necessary PID drive
int getPIDdrive(int error, int prev_error, double *errorI)
```

```
{
	double aux_errorI = *errorI;
	double drvP, drvI, drvD, drvPID;

	//Proportional Drive
	drvP = (double)(error * KP);

	//the Integral Drive will have a contribution only when the error is not very
large
	//The integral will control the DC error
	if (aux_errorI < IMAX_LIM && aux_errorI > -IMAX_LIM && error != 0)
	{
		aux_errorI += error;
	}
	else
	{
		aux_errorI = 0;
	}
	//The integral error is limited
	if (aux_errorI > (double)(5 / KI))
	{
		aux_errorI = (double)(5 / KI);
	}
	if (aux_errorI < -(double)(5 / KI))
	{
		aux_errorI = -(double)(5 / KI);
	}
	*errorI = aux_errorI;
	//Integral Drive
	drvI = (double)(KI * aux_errorI);

	//if the error is 0, then the derivative drive will not have a contribution
	if (error == 0)
	{
		drvD = 0;
	}
	else
	{
		//Derivative Drive, acting on past errors
		drvD = (double)((error - prev_error) * KD);
	}

	//The final necessary drive
	drvPID = drvP + drvI + drvD;
	if (drvPID < 0)
		drvPID *= -1;
	if ((int)(drvPID) > MAX_PWM)
		 drvPID = MAX_PWM;

	return ((int)(drvPID));
}
```

3. Main program

```
/*
 * Test1.c
 *
 * Created: 3/22/2020 12:29:54 PM
 * Author : Bogdan
 */

#include "LCD.h"
#include "Motor.h"
#include <math.h>
#include <stdlib.h>

volatile int motorPosition = 0;
volatile int pinionPosition = 0;
volatile double velocity = 0;
volatile double prev_velocity = 0;
volatile double acceleration = 0;
volatile unsigned int count10us = 1;
volatile int refPosition = 0;
volatile int refPosition_select = 0;

int main(void)
{
	setupPorts();
	LCD_Setup();
	LCD_InitialDisplay();
	MotorSetup();

	int error = 0;
	int prev_error = 0;
	double errorI = 0;
	int drvPID = 0;

    while (1)
    {
		prev_error = error;
		error = getError(refPosition, motorPosition);
		MotorSetDirection(getDirection(refPosition, pinionPosition));
		drvPID = getPIDdrive(error, prev_error, &errorI);
		MotorSetDuty(drvPID);
		LCD_DisplayVel(velocity);
		LCD_DisplayAcc(acceleration);
		LCD_DisplayPos(pinionPosition);
		LCD_DisplayRef(refPosition_select);
    }
}

//Pin change interrupt 0, on PORTB0, PORTB1, PORTB2
ISR (PCINT0_vect)
{
	//REF+ is pressed
	if (!(PINB & (1 << PINB0)))
	{
		//Limit the reference to the maximum position of the pinion
		if (refPosition_select == RACK_TEETH)
		{
```

```
                        refPosition_select = RACK_TEETH;
                }
                else
                {
                        refPosition_select++;
                }
        }
        //REF- is pressed
        if (!(PINB & (1 << PINB1)))
        {
                //Limit to the minimum position
                if (refPosition_select == 0)
                {
                        refPosition_select = 0;
                }
                else
                {
                        refPosition_select--;
                }
        }
        //The desired position is confirmed
        if (!(PINB & (1 << PINB2)))
        {
                refPosition = refPosition_select;
                startTimer0();
        }
}

//Timer1 interrupt, generated at every 10 us
ISR (TIMER1_COMPA_vect)
{
        count10us++;
}

//External interrupt 0
ISR (INT0_vect)
{
        //if a rising edge is detected when the other signal is low means a Motor position
increment
        if (!(PIND & (1 << PIND3)))
        {
                stopTimer1();
                motorPosition++;
                //set the Pinion position
                pinionPosition = motorPosition / PINION_CONSTANT;
                //calculates the velocity based on the rate of change in position
                prev_velocity = velocity;
                velocity = (double)(VELOCITY_CONST / count10us);
                //calculates the acceleration, based on the current velocity and and
previous velocity
                acceleration = (double)(((velocity * 100000 - prev_velocity * 100000) /
count10us));
                //reset the counter
                count10us = 0;
                startTimer1();
        }
}
```

```
//External interrupt 1
ISR (INT1_vect)
{
	//if a rising edge is detected when the other signal is low means a Motor position
decrement
	if (!(PIND & (1 << PIND2)))
	{
		stopTimer1();
		motorPosition--;
		//set the Pinion position
		pinionPosition = motorPosition / PINION_CONSTANT;
	//calculates the velocity based on the rate of change in position
		prev_velocity = velocity;
		velocity = (double)(VELOCITY_CONST / count10us);
		//calculates the acceleration, based on the current velocity and and
previous velocity
		acceleration = (double)(((velocity * 100000 - prev_velocity * 100000) /
count10us));
		//reset the counter
		count10us = 0;
		startTimer1();
	}
}
```