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:

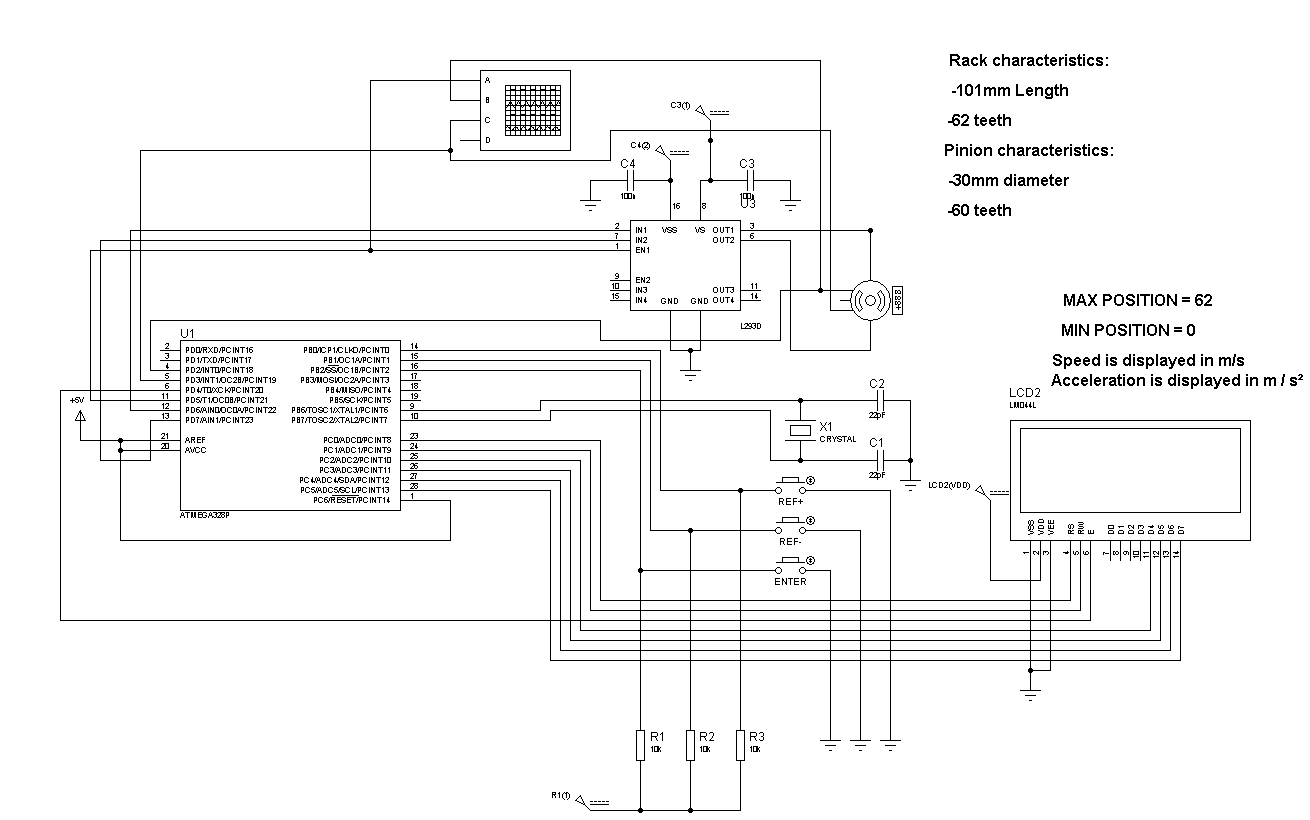
-The program is written in C and was developed using Atmel Studio. The source code has three modules: LCD module, Motor module and Main program.

-The entire system is simulated using Proteus.

-Link to the project which includes a video demonstration:

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

# Schematic:



# 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\_ \*/

/\*

\* 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();

}

}