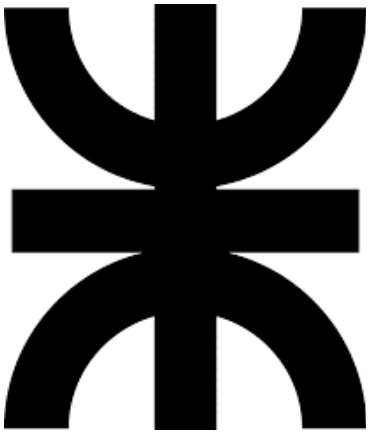
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA

INGENIERÍA ELECTRÓNICA



INFORMÁTICA II

Brazo Robótico tri axial

TRABAJO PRÁCTICO GRUPO N°2

DOCENTES Herencia Juan José,
Martinez José Luis.

COMISIÓN 2R2

ALUMNOS		
	De Robles Agustin	80499
	Giorgis Ezequiel	91661
	Luna Joaquín	75934

Córdoba, 14 de noviembre de 2022

CONTENIDO

1. Resumen

2. Introducción

El presente documento tiene como fin exponer, desarrollar y explicar la codificación realizada en la construcción de un brazo robótico triaxial. Para la elección del tema nos inspiramos en las lineas de producción y en su constante desarrollo e importancia. Particularmente en los brazos robóticos que, por una cuestión de reducción de costos y mayor eficacia en su trabajo, están cada vez mas presentes en las fabricas.

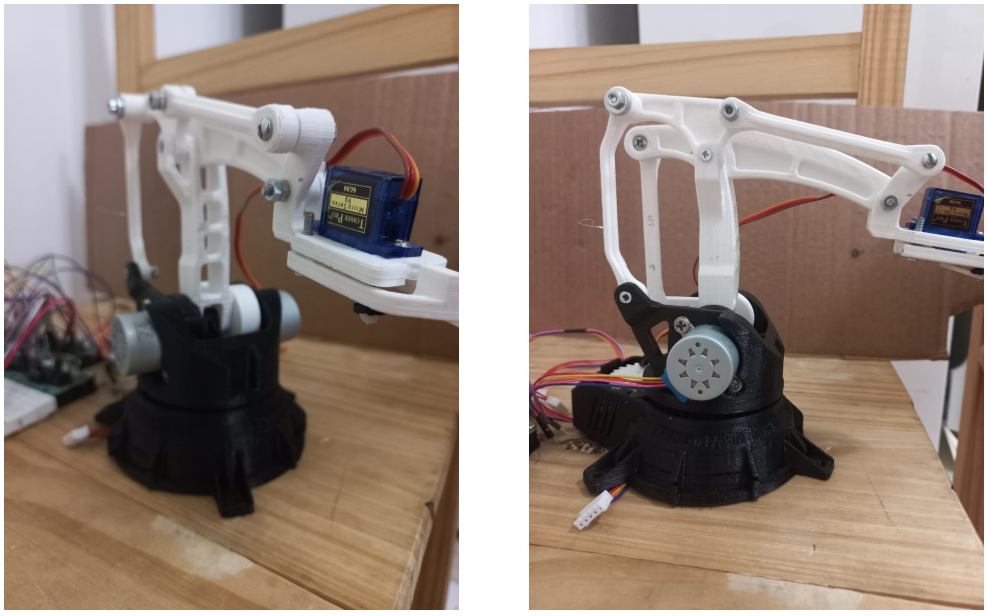
Nuestro objetivo fue construir un brazo robótico que pueda recibir instrucciones por computadora con la finalidad de moverse en tres ejes y abrir y cerrar una pinza con presicion. En un futuro tambien podría adaptase para categorizar un producto según su color, su forma o tamaño.

2.1. Descripción general

La construcción mecánica del brazo robótico esta basada en impresión 3D en material PLA. De esta forma, tres motores paso a paso se encargan de mover, mediante engranajes, el brazo en los tres ejes. Como cabezal se implemento una pinza accionada por un servo motor.

Como microcontrolador se utilizo el Atmega328p, además de tres drives Unl2003 encargados del control de los motores paso a paso.

Tanto el código para el microcontrolador, como el programa de consola para la PC, fue implementado en C en su totalidad. Sin embargo, la interfaz gráfica para la PC, fue desarrollada en C++ por medio del entorno de programación QtCreator.



(a) (b)

Figura 1: Imagen del brazo robótico

3. Hardware

En el apartado de hardware se destaca, como bien dijimos, el uso de tres motores paso a paso 28BYJ-48 cuyas especificaciones son:

- Tensión nominal de entre 5V y 12 V.
- 4 Fases.
- Resistencia 50 Ω.
- Par motor de 34 Newton / metro (aprox. 0,34 Kg por cm).
- Consumo de unos 55 mA.
- 8 pasos por vuelta.
- Reductora de 1 / 64.
- Unipolar

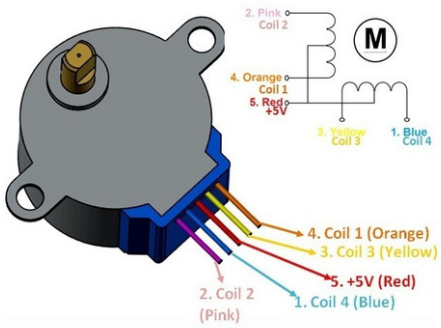
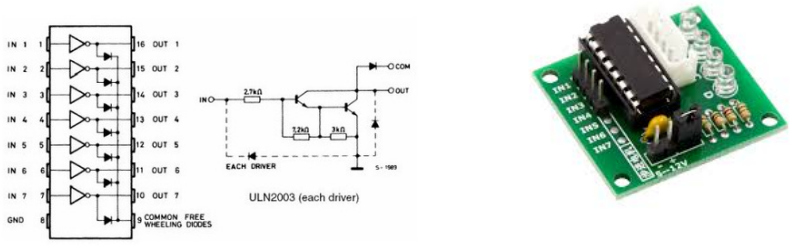


Figura 2: Motor $28BYJ_48$

Para controlar estos motores, se emplearón drives UNL2003 constituido por dos transistores en configuración Darlington.



(a) (b)

Figura 3: Driver ULN2003

Para la apertura y cierre de la pinza, se utilizó un servo motor Sg90; el cual posee un torque de 9 gramos.

Figura 4: Servo Sg90

4. Proceso de compilado y flash

Para la compilación del código correspondiente al microcontrolador, como así también al de la PC, se emplearon los siguientes compiladores y programas/utilidades de consola:

- gcc
- g++
- avr-dude
- avr-copy
- avr-size

Todos ellos implementados en la herramienta de gestión de dependencias Make.

5. Software Microcontrolador

5.1. Función Main

Dentro del código se incluyen todos los archivos de cabecera dentro de un único llamado allInc.h. Este último contiene, entre otras cosas, las funciones SetInOut, pwm init y las relacionadas a la Uart.

Con las funciones SetInOut se inicializan los pines del arduino, con UART init se configura la velocidad para la uart y la comunicación serial y con pwm init se inicializa el pwm y se setea el valor mínimo (0) y máximo (180) del servomotor para la amplitud de la apertura de la pinza. En el bucle infinito, la función UART gets lee el puerto de serie hasta que haya un salto de línea, almacenando el dato en la variable line para luego traducirlo con la función parse en comandos de movimiento, devolviendo error u ok según corresponda.

```
1 #include <util/delay.h>
2 #include "../inc/allInc.h"
3
4 int main(void) {
5   SetInOut();
```

```
  UART_init();
  pwm_init();
  sei();

  servo_set(0,180);

  char line[80];

  UART_puts("OK\n");
  for(;;){
    UART_gets(line);
    if(parse(line, execLine))
      UART_puts("[ERROR]\n");
    else
      UART_puts("OK\n");
  }

  return 0;
}
```

5.2. UART

Debido a las necesidades del presente proyecto, se optó por la utilización de una comunicación en serie asíncrona de tipo half-duplex bajo el estándar RS-232 con una configuración de trama 115200/8N1. Las características de dicha comunicación entre dispositivos se detallan a continuación:

Comunicación serie

La comunicación serie o serial transmite un único bit a la vez de forma secuencial, es decir, un bit tras otro, uno a la vez. Aunque es más lenta que una comunicación paralela, tiene la ventaja de que físicamente requiere de una sola conexión (un solo cable). Además, tiene un límite de distancia mucho mayor frente a una comunicación en paralelo.

Asíncrona

En la comunicación asíncrona, la sincronización se restablece con la transmisión de cada carácter mediante el uso de bits de inicio y parada que, dependiendo de la tecnología utilizada, puede haber 1, 1,5 o 2 bits de parada.

Half-Duplex

De manera similar, cuando dos dispositivos de comunicación de datos se comunican en un entorno Half-Duplex, un dispositivo debe ser el transmisor y el otro el receptor. Para permitir la comunicación bidireccional, periódicamente tienen que invertir roles.

Estándar RS-232

La sección de características eléctricas del estándar RS-232 incluye especificaciones sobre los niveles de tensión, tasa de cambio de niveles de las señales e impedancia de

la línea. El estándar RS-232 original se definió en 1962, previo a los días de la lógica TTL (Transistor-Transistor Logic), por lo que no es de extrañar que no utilice niveles lógicos de 5 voltios. En cambio, se define un nivel alto de salida con tensiones entre +5 y +15 voltios y un nivel bajo entre –5 y –15 voltios. Los niveles lógicos de entrada se definieron para proporcionar un margen de ruido de 2 voltios, por lo tanto un nivel alto para la entrada debe estar comprendido entre +3 a +15 voltios y un nivel bajo entre –3 a –15 voltios. En el actual proyecto, por incidencia directa del microcontrolador Atmega328, se emplearon salidas de +5V para niveles lógicos altos.

Velocidad de transmisión

Indica el número de bits por segundo. En nuestro caso 115200 por defecto.

Trama 8n1

El número de bits de datos

Refiere a la cantidad de bits (word) en la transmisión. En nuestro caso 8 bits.

El número de bits de stop

Utilizado para indicar el fin de la comunicación de un paquete. Mientras más bits de paro se usen, mayor será la tolerancia a la sincronía de los relojes de los dispositivos involucrados, sin embargo, la transmisión será más lenta. Para este proyecto se utilizó un bit de stop.

Y si cuenta con bit de paridad

El bit de paridad es una forma sencilla de verificar si hay errores en la transmisión serial. Para nuestro contexto no fue necesario definir bits de paridad, puesto que la complejidad de los datos enviados y recibidos no era elevada.

Finalmente para configurar, con las características mencionadas con anterioridad, la UART del microcontrolador Atmega328 se modificaron los registros correspondientes según la hoja de datos del fabricante y según la siguiente fórmula para determinar el registro de UART Baud Rate Register (UBRR0).

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous normal mode (U2Xn = 0)	$BAUD = \frac{f_{osc}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous double speed mode (U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{8BAUD} - 1$
Synchronous master mode	$BAUD = \frac{f_{osc}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps)

BAUD	Baud rate (in bits per second, bps)
f_{osc}	System oscillator clock frequency
UBRRn	Contents of the UBRRnH and UBRRnL registers, (0-4095)

Figura 5: Tabla para calcular el registro de Baud Rate

```
void UART_init(void) {
#if BAUD < 57600
  uint16_t UBRR0_value = ((F_CPU / (8L * BAUD)) - 1) / 2;
  UCSR0A &= ~_BV(U2X0);  // baud doubler off
#else
  uint16_t UBRR0_value = ((F_CPU / (4L * BAUD)) - 1) / 2;
  UCSR0A |= _BV(U2X0);  // baud doubler on
#endif
  UBRR0H = (uint8_t) UBRR0_value >> 8;
  UBRR0L = (uint8_t) UBRR0_value;

  UCSR0B |= _BV(RXEN0); // rx on
  UCSR0B |= _BV(TXEN0); // tx on
  UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // trama 8N1
}
```

5.3. Funciones parse y execLine

Nos permite ingresar una cadena de caracteres en donde discrimina los espacios en blanco. Posteriormente mediante la función strToUpper nos permite hacer la conversión a una letra Mayúscula. Esto nos permite mayor tolerancia al momento de ingresar en qué eje se ha de desplazar el brazo, tomando en cuenta que si se ingresa una letra Minúscula hará la conversión correspondiente y permitirá realizar la acción de los motores paso a paso. En dado caso en que se ingrese un carácter S ejecutará la acción de la función execLine que hará accionar el servo.

Dado el caso en el que se ingrese x,y,z por teclado y también se ingrese un valor por ejemplo 100, la letra indicará hacia que eje debe hacer el movimiento el brazo y el valor indicará la cantidad de paso(step) que debe hacer en dicha dirección.

El programa contempla el sentido en que debe desplazarse el brazo, dado que puede realizar el desplazamiento en ambos sentidos.

5.4. Control de motores y servo motores

Para conseguir esta secuencia se define, primeramente, un arreglo de tipo uint8 t (unsigned char) constante donde serán almacenados los pines y el puerto correspondiente a cada entrada del driver Estos serán pasados, junto al número de pasos a ejecutar, por referencia y copia respectivamente, a la función moveAxisRelative. Esta define un contador para indicar el paso actual, por lo que varía entre 0 y 3. Por último, se hace uso de un arreglo de funciones, doStepHorario, para llamar a la función del paso que corresponda. Función que es la encargada de apagar y energizar las bobinas del motor por medio del registro de 8 bits correspondiente al puerto, y con el uso de los operadores a nivel de bit y sentencias correspondientes para apagar y encender respectivamente.

Para la apertura y cierre del servomotor, se hizo uso de los timers del propio Atmega328 para generar una señal de pulso modulado (pwm). De este modo, haciendo variar el duty cycle de la señal, conseguimos que el servo motor se posición según lo deseado.

```
void primerStep(volatile uint8_t* port, const uint8_t *axisPin){
  *port &= ~_BV(axisPin[1]);
  *port &= (~_BV(axisPin[1]) || ~_BV(axisPin[2]) || ~_BV(axisPin[3]) ||
    ~_BV(axisPin[4]));
  *port |= _BV(axisPin[1]);
}
```

6. Software PC

6.1. Comunicación serial

Para llevar a cabo una comunicación serial exitosa, ambos dispositivos deben tener la misma configuración. Por ello, para implementar la configuración descrita en la sección (5.2) se abrió el dispositivo tratado como archivo, para luego settear sus características por medio de la estructura términos.

```
  if (tcgetattr(fd, ttyp) != 0) {
    printf("ERROR: tcgetattr\n");
    return -1;
  }

  cfsetospeed(ttyp, baudrate);
  cfsetispeed(ttyp, baudrate);

  ttyp->c_cflag = (ttyp->c_cflag & ~CSIZE) | CS8; /* 8 data bits (8) */
  ttyp->c_cflag &= ~(PARENB | PARODD);            /* no parity   (N) */
  ttyp->c_cflag &= ~CSTOPB;                       /* 1 stop bit  (1) */
  ttyp->c_cflag |= (CLOCAL | CREAD); /* ignore modem status lines, and
   */
```

6.2. Función Main

Luego de abrir como un archivo y configurar el microcontrolador, se inicia un ciclo for infinito que permitirá enviar las correspondientes instrucciones al brazo: direcciones y sentido en las que se desee que se realice el desplazamiento como así también la

cantidad de pasos (step) que deben hacer en dicha dirección. Todo esto últimos datos son enviados en forma de cadena de caracteres al microcontrolador.

```
char buf[80] = {0};
for (;;) {
  read(fd, &buf, 8);
  tcdrain(fd);
  printf("[MICRO] %s", buf);
  fgets(buf, 79, stdin);
  tcdrain(fd);
  write(fd, buf, bitsUntilc(buf, '\n'));
  tcdrain(fd);
  sleep(1);
}
```

6.3. GUI

El entorno gráfico cuenta con 13 botones en total, 12 para mover el brazo en los ejes x,y,z; y uno para accionar la pinza.

En el código se inicializa el id del producto y el del proveedor del arduino uno. Se procede a reconocer los puertos disponibles y comparar el identificador del producto y el del proveedor conectados con los previamente seteados. Si coinciden, en la variable arduino uno port name se copia la información referente al puerto de serie, que se utilizará luego para inicializar el nombre del puerto.

Se procede a verificar que, dentro de la comunicación serial, el baudrate, los bits, la paridad, etc. sean los requeridos para seguir con el proceso. Se espera por un dato de entrada, dicha información se lee y se depura. En caso de no poder hacerlo se imprime un error.

Al presionar un botón, ejemplo -x se verifica que se pueda escribir el puerto de serie y, si es así, se envía un valor, en este caso, de -100.

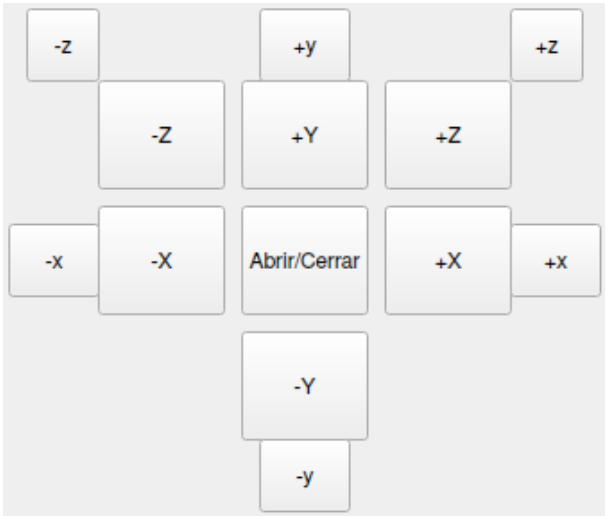


Figura 6: Implementación interfaz gráfica

7. Conclusiones

La construcción final del brazo robótico demuestra de por sí la capacidad del quienes subscriben de programar softwares de bajo nivel necesarios para las industrias de hoy en día. Además, gracias al paradigma de programación estructurada por el que se opto, se logro crear un código, no solo flexible, sino también reutilizable y adaptable según la envergadura y necesidades de cada proceso industrial.

8. Referencias

[1] Atmel Corporation. *Atmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*, 2015.

[2] P.J. Deitel H.M. Deitel. *Como programar en C/C++*. Prentice Hall Hispanoamericana, S.A., second edition, 1995.

[3] Gonzalo Perez Paina. Unidad c̈ontrol de periféricosc̈omunicación serie entre pc y placa arduino", 2022.

[4] Joe Pardue. *C Programming For Microcontrollers*. Smiley Micros, 2005.

[5] Paul D. Smith Richard M. Stallman, Roland McGrath. *GNU Make Manual*. Free Software Foundation, 0.75 edition, 2020.

[6] Bjarne Stroustrup. *The C++ Programming Language*. Pearson Education, Inc., fourth edition, 2013.