Carro-Torreta Móvil con el ATMega328P

Mario Betancourt, *23440, UVG,*

I. Introducción

En la edición 2025 del curso de *Programación de Microcontroladores* de la Universidad del Valle de Guatemala, los estudiantes aprendieron a programar el microcontrolador ATMega328P usando códigos en lenguaje ensamblador y C. En el transcurso del semestre que duró, se exploraron tópicos concernientes al uso de pines de propósito general (GPIOs), manejo de información en las memorias del microcontrolador (RAM y ROM), interrupciones, técnicas para administrar múltiples dispositivos (*multiplexado*), temporizadores, generadores de señales PWM, el módulo ADC, el almacenamiento de información en la EEPROM integrada y el uso de AdafruitIO para generar interfaces gráficas.

Para finalizar, los estudiantes realizaron un proyecto libre, siempre y cuando cumpliera con lo siguiente: *usar cuatro entradas analógicas, controlar cuatro servomotores o motores DC con las entradas recibidas, permitir la transmisión de datos con una computadora a través de comunicación serial (USART), almacenar posiciones en la memoria EEPROM y utilizar una interfaz gráfica en AdafruitIO*. Mario Betancourt, el creador de este proyecto, construyó un *carro con una torreta*, usando materiales disponibles en tiendas locales. En este reporte se describe el Hardware y Software del proyecto, para que este pueda replicarse con facilidad.

Mario βetancourt
Mayo 23, 2025

II. Configuraciones de Módulos - Cálculos

A. Modo Fast PWM en Timer 1 - Servomotores

Entre los tres temporizadores del ATMega328P, el Timer1 es el único que tiene un contador de 16 bits. Esto le permite desbordarse en intervalos de tiempo más grandes, así como en intervalos de tiempo más exactos. En el programa del microcontrolador a bordo del carro, el Timer1 se configuró en el modo 14, el modo Fast PWM con TOP establecido en ICR1. De esta forma, fue posible establecer la frecuencia de la señal PWM con el valor de ICR1 y controlar los ciclos de trabajo de los generadores de señal OC1A y OC1B, con los registros OCR1A y OCR1B.

Para controlar dos servomotores SG90 de 180°, es necesario generar una señal PWM con un periodo de 20 ms (O, de forma recíproca, una frecuencia de 50 Hz), con un tiempo en alto de 1-2 ms. Esto es lo recomendable por los proveedores y se puede comprobar en las hojas de datos de estos dispositivos. El prescaler mínimo necesario para generar dicha señal se encuentra con la siguiente ecuación:

$$prescaler \geq \frac{t_{delay} \times f_{clk}}{2^n}$$

Tomando $t_{delay} = 20 \times 10^{-3}\ s$, $f_{clk} = 16 \times 10^6\ Hz$, y $n = 16$ y sustituyendo valores:

$$prescaler \geq \frac{(20 \times 10^{-3}\ s) \times (16 \times 10^6)}{2^{16}} = 4.88$$

El divisor de prescaler disponible inmediatamente superior a este número es 8. Por lo tanto, *se usa un prescaler de 8 para el Timer1* ($prescaler = 1$) y se habilita el bit CS11 en el registro. Con este número se puede resolver la siguiente ecuación para calcular el valor de TOP:

$$f_{PWM} = \frac{f_{clk}}{prescaler \times (1 + TOP)}$$

Despejando para TOP:

$$TOP = \frac{f_{clk}}{prescaler \times f_{PWM}} - 1$$

Sustituyendo valores:

$$TOP = \frac{(16 \times 10^6\ Hz)}{8 \times (50\ Hz)} - 1 = 39999$$

Por consiguiente, se debe configurar $ICR1 = 39999$. Finalmente se determinan los valores teóricos de OCR1x (Donde x es A o B) para generar una señal con tiempos en alto de 1 ms (Duty Cycle = 5 %) a 2 ms (Duty Cycle = 5 %). En ambos casos se utiliza la siguiente ecuación:

$$DC = \frac{OCR1x + 1}{2^n} \times 100\%$$

Donde DC es el porcentaje de ciclo de trabajo ($0 \leq DC \leq 100$). Despejando para $OCR1x$:

$$OCR1x = \frac{2^n \times DC}{100\%} - 1$$

1) Duty Cycle del 5%

$$OCR1x = \frac{2^{16} \times 5}{100} - 1 = 3275.8 \approx 3276$$

2) Duty Cycle del 10%

$$OCR1x = \frac{2^{16} \times 10}{100} - 1 = 6552.6 \approx 6553$$

Para ajustar el valor del ciclo de trabajo con un registro de 8 bits se realiza el un mapeo para ajustar la posición de los servomotores entre 0° y 180°. Este se basa en una relación lineal entre el número de ocho bits, sea este $value$ y $OCR1x$. La pendiente es:

$$pendiente = \frac{\Delta OCR1x}{\Delta value} = \frac{6553 - 3276}{255 - 0} = 3277/255$$

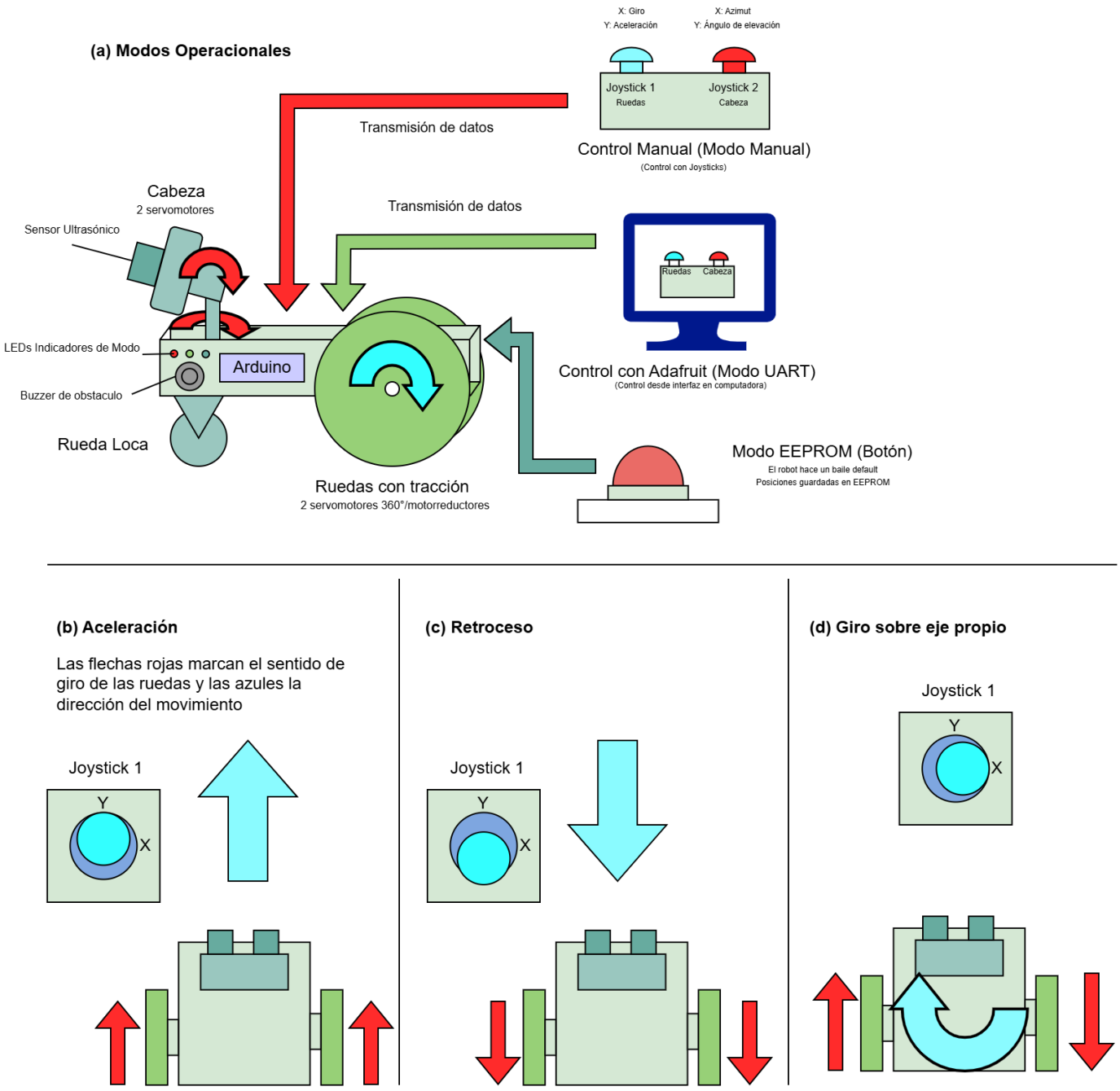

Fig. 1. Concepto de Operaciones (En la versión final se descartaron el sensor ultrasónico y, por obvias razones, el buzzer de obstáculo).

Por lo tanto se podría utilizar la siguiente relación para establecer el ciclo de trabajo:

$$OCR1x = 3276 + \frac{3277}{255} value$$

Es importante notar que esta ecuación permite girar el eje de un servomotor entre los ángulos mínimo y máximo permisibles. Esto podría resultar inconveniente cuando el espacio o la fuerza máxima que deben ejercer los servomotores es limitada. Por esta razón se puede reducir o redondear los valores de esta ecuación en el mismo orden de magnitud y obtener resultados igualmente satisfactorios. En la versión final del proyecto, la siguiente ecuación probó ser adecuada para controlar los servomotores sin llevarlos a posiciones donde pudieran dañarse:

$$OCR1x = (value * 4000)/255 + 1000$$

Con esta expresión el valor mínimo de OCR1x es 1000 y su valor máximo es 5000.

B. Modo Fast PWM en Timer 0 - Motorreductores

El Timer0 fue utilizado para generar dos señales PWM para controlar los motorreductores que impulsan al carro en sus ruedas traseras. Si bien no existe una frecuencia recomendada para la transmisión de potencia a cargas inductivas, el factor

decisivo bajo el cual se seleccionó este parámetro fue el *sonido que producen los motores*. Cuando la frecuencia de la señal PWM entra dentro del espectro audible por el oido humano, es posible escuchar un timbre de frecuencia constante aún cuando los ejes de los motores *no están rotando*. Por esta razón, si la frecuencia está en el extremo superior del espectro audible, los motores podían producir timbres agudos que, si bien no son dañinos por intervalos de tiempo cortos, pueden resultar irritantes para quienes trabajan con los motores.

Aunque la respuesta más conveniente es *siempre utilizar una frecuencia fuera del rango audible*, en este proyecto se hizo un aprovechamiento del sonido para demostrar el funcionamiento de los motores en su presentación final. Debido a una mala elección del sistema de potencia, las baterías escogidas para alimentar los motorreductores resultaron ser insuficientes para que sus ejes roten. Sin embargo, al probar el sistema completo con los generadores de señal conectados a los motores, fue posible reconocer que al mover el Joystick que controla estos dispositivos, el volumen del sonido producido podía aumentar o disminuir. ¡La potencia de los motores sí estaba siendo controlada por la señal PWM! El problema, es que la potencia era demasiada como para suplirla con cuatro baterías AA.

Para aprovechar este fenómeno antes de reemplazar la fuente de voltaje, se escogió una frecuencia cercana a la de una nota musical. Por limitaciones de Hardware se generaron señales de 244 Hz, cercanas a la nota D#4 (Re sostenido cuarta). De esta forma, aunque los motores no giraran, se podría comprobar que están en funcionamiento dado que producen un sonido que no resulta molesto.

A continuación se presentan los cálculos del prescaler en TCCR0B y el valor inicial del contador TCNT0. Se asume una frecuencia de sistema $f_{cpu} = 16\ MHz$.

Recordando que el periodo de delay es igual al recíproco de la frecuencia, se calcula el prescaler mínimo usando

$$prescaler \geq \frac{f_{clk}}{2^n \times f_{signal}}$$

Sustituyendo valores

$$prescaler \geq \frac{16 \times\ 10^6 Hz}{2^8 \times 244\ Hz}$$

$$prescaler \geq 256.14$$

El prescaler debía ser mayor o igual a 256. Afortunadamente, ya que la exactitud no es clave para la transmisión de potencia a los servomotores, es permisible utilizar un valor de 256 por la diferencia mínima. Este cálculo se hizo asumiendo $TCNT0 = 0$, lo que facilita el reinicio del contador después de un overflow.

C. Módulo ADC - Joysticks

El módulo convertidor de Analógico-a-Digital (por sus siglas en inglés, *ADC*) se utilizó para recibir cuatro entradas analógicas de dos joysticks de potenciómetros, las cuales controlan los cuatro motores a bordo. Por el Teorema de Nyquist la frecuencia de muestreo debería ser igual al doble de la frecuencia máxima de las señales de entrada, para poder reconstruir la señal analógica completamente. Debido a que los controles eran accionados manualmente, es permisible utiliza cualquier frecuencia de muestreo para la cual su recíproco (El periodo entre conversiones completas) sea significativamente menor al tiempo de reacción de una persona. Para una frecuencia del reloj de sistema de 16 MHz, un prescaler de 64 probó ser un valor funcional, aunque prácticamente cualquier prescaler disponible hubiera funcionado igualmente bien. No hace falta fundamentar esta elección en cálculos.

A pesar de que la elección del prescaler del ADC es arbitraria, es recomendable considerar el *timing* entre las señales de los módulos del microcontrolador para recibir y reiniciar el convertidor efectivamente. Si la llegada de las señales está mal organizada, es posible que ocurra una *condición de carrera*, y el orden de las instrucciones ejecutadas no sea el esperado por el programador. En el desarrollo del proyecto hubieron conflictos entre las interrupciones del ADC y el TIMER0 por problemas de concurrencia.

Fuera de las cuestiones del Timing, las otras configuraciones relevantes del módulo ADC en este proyecto son (1) que el resultado de la conversión está ajustado a la izquierda (ADLAR = 1) y (2) el Trigger libre está activado (ADCSRB = 0x00). En partícular, con esta última configuración, el módulo ADC empieza una conversión inmediatamente después de haber terminado otra conversión. Aunque esto consuma un poder de procesamiento mayor, esta elección evita que el módulo ADC dependa de otros periféricos y por la misma razón, las conversiones no ocurran en el orden o el tiempo esperado. Esto puede ocurrir al escoger el desbordamiento de un temporizador como el trigger del ADC.

Como se puede observar en el diagrama de flujo 1 el canal del ADC se puede configurar a través de la función *ADC set channel* (Establecer canal en ADC), donde se aplica una máscara al registro ADMUX para conservar sus primeros bits y modificar los cuatro últimos de acuerdo a una entrada de 8 bits. Esta función es *clave* para lograr un multiplexado de canales, como se mostrará más adelante en la sección de algoritmos.

D. Configuraciones de UART

Finalmente se presentan los cálculos utilizados para establecer una comunicación serial duplex entre el microcontrolador en el carro y una computadora portatil. Para ello se utiliza el módulo USART, configurado para establecer una comunicación asíncrona normal (U2Xn = 0) Se utilizó un *baudrate* de 9600. El valor de UBRR0 se calculó utilizando la siguiente ecuación

$$UBRR0 = \frac{f_{cpu}}{16 \times BAUD} - 1$$

sustituyendo valores:

$$UBRR0 = \frac{16 \times 10^6\ Hz}{16 \times 9600\ Hz} - 1 = 103$$

Este resultado (Originalmente de 103.166) se redondea al valor entero más cercano. Debido a este redondeo la tasa de baudios establecida tiene un error respecto a los 9600 baudios exactos. Con UBRR0 = 103 el *baudrate* esperado es:

$$BAUD = \frac{f_{cpu}}{16(UBRR0 + 1)}$$

sustituyendo valores

$$BAUD = \frac{16 \times 10^6 Hz}{16(103 + 1)} = 9616$$

El porcentaje de error de transmisión es:

$$\%error = \frac{9616 - 9600}{9600} \times 100\% = 0.16\%$$

Este es un valor permisible y aceptable para una comunicación estable.

Debido a que es necesario que el microcontrolador reciba y envíe datos, al igual que la computadora, se habilitaron el *receptor* y el *transmisor* del microcontrolador, así como las interrupciones por recepción de datos. Adicionalmente, se escogió un framing de 8 bits, sin paridad con un bit de terminación.

III. Algoritmos y Diagramas de Flujo

En esta sección se presentan los diagramas de flujo de los códigos empleados en esta solución. Aquí se incluyen códigos que no implican la inicialización de periféricos.

A. Multiplexado de Canal de ADC

Debido a que es necesario hacer una revisión continua de las entradas analógicas, el canal del módulo ADC debe cambiarse continuamente en una secuencia definida. Por la similitud existente entre este procedimiento y la técnica de multiplexado que se aprendió para displays de siete segmentos, se le denomina **multiplexado de canal de ADC**. Para cambiar el canal del ADC rápidamente, se elaboró la función *adc_set_channel()*, previamente mencionada en la sección de las configuraciones del módulo del ADC. Como se puede apreciar en la figura 2 el canal se escoge aplicando una máscara sobre los últimos cuatro bits de ADMUX. Los canales del ADC están numerados en orden decimal, por lo que al ingresar el número decimal de un canal lo activa inmediatamente.

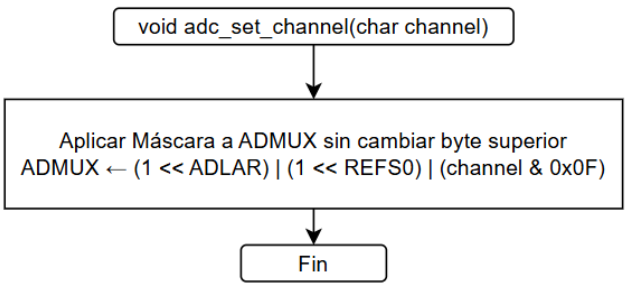

Fig. 2. Cambio de canal de ADC

Esta función se utilizó dentro de la rutina de interrupción por conversión completa *ISR(ADC_vect)* para construir una especie de *máquina de estados finitos* (Figura 3). Se utiliza una sentencia switch-case y una variable global *current_channel* para que, al entrar al caso de un canal, en la siguiente interrupción se lea el canal inmediatamente superior. Cuando el contador llega al canal 3 este se reinicia y el ciclo continua.

B. Comunicación Serial

Para la comunicación serial se elaboraron funciones que transmitan datos hacia el receptor (En este caso, la computadora) en forma de caracteres y cadenas de caracteres. En la solución, estas funciones se utilizan únicamente para escribir en la terminal de la computadora, para facilitar la prueba de los módulos fuera de la interfaz gráfica.

Una observación interesante es que la función para enviar cadenas de caracteres utiliza un argumento definido por un *puntero*. Este apunta en la dirección en la que se guardan los datos, y al incrementar su valor en 1, el puntero se mueve en la dirección inmediatamente superior. Esto asemeja a lo realizado con direccionamiento indirecto en lenguaje ensamblador.

En el código principal la interrupción por recepción de datos se utiliza para recibir caracteres de frames multibyte de 5 bytes cada uno. Para reconocer el inicio y el fin de un frame, cada frame debe comenzar con un *caracter de inicio* ('0') y terminar con un *caracter de terminación* ('Z'). Los dos bytes que proceden al byte de inicio son dos *bytes de instrucciones* y codifican una acción para que el microcontrolador la interprete. El cuarto byte del frame es un *byte de datos* y puede contener, por ejemplo, el valor que se cargará para establecer la posición de un servomotor o la rapidez de un motorreductor. En general el byte de datos equivale a una entrada analógica en el modo manual. Para resumir lo antes mencionado, la estructura de los frames es la siguiente: [INICIO] → [INSTRUCCIÓN 1] → [INSTRUCCIÓN 2] → [DATO] → [FIN].

Para reconocer un frame, la interrupción por recepción de datos en USART revisa si el dato recibido es el caracter de inicio, después de haber comprobado si una bandera que indica si la bandera de frame listo está inactiva. Si el byte recibido es el byte de fin y el índice de recepción es la longitud del frame, entonces el frame termina y se levanta la bandera frame_ready. Si no es ninguno de los dos y el índice de recepción es menor que la longitud de la cadena, se guardan los datos en un arreglo de caracteres.

C. Puente H y movimiento de motorreductores

Para permitir giros en dos direcciones, se utilizó un módulo puente-H L298N, con regulación de voltaje y diodos de protección incorporados. Para establecer la dirección de giro de un motor, es necesario colocar un uno lógico en una entrada y un cero lógico en la otra. Si ambas entradas tienen el mismo valor lógico, el motor se detiene. Esto se resume mejor en la figura 7.

El prototipo construido utiliza dos ruedas impulsadas y una rueda loca (Una rueda con dirección libre). A este tipo de sistema de ruedas se le conoce como **ruedas diferenciales**. Si bien ambas ruedas están alineadas, al hacerlas girar con velocidades angulares distintas, es posible hacer que el vehículo gire alrededor de un centro de rotación externo o interno, determinado por la *diferencia* entre las velocidades de las ruedas. En la solución construida, este sistema se realizó de

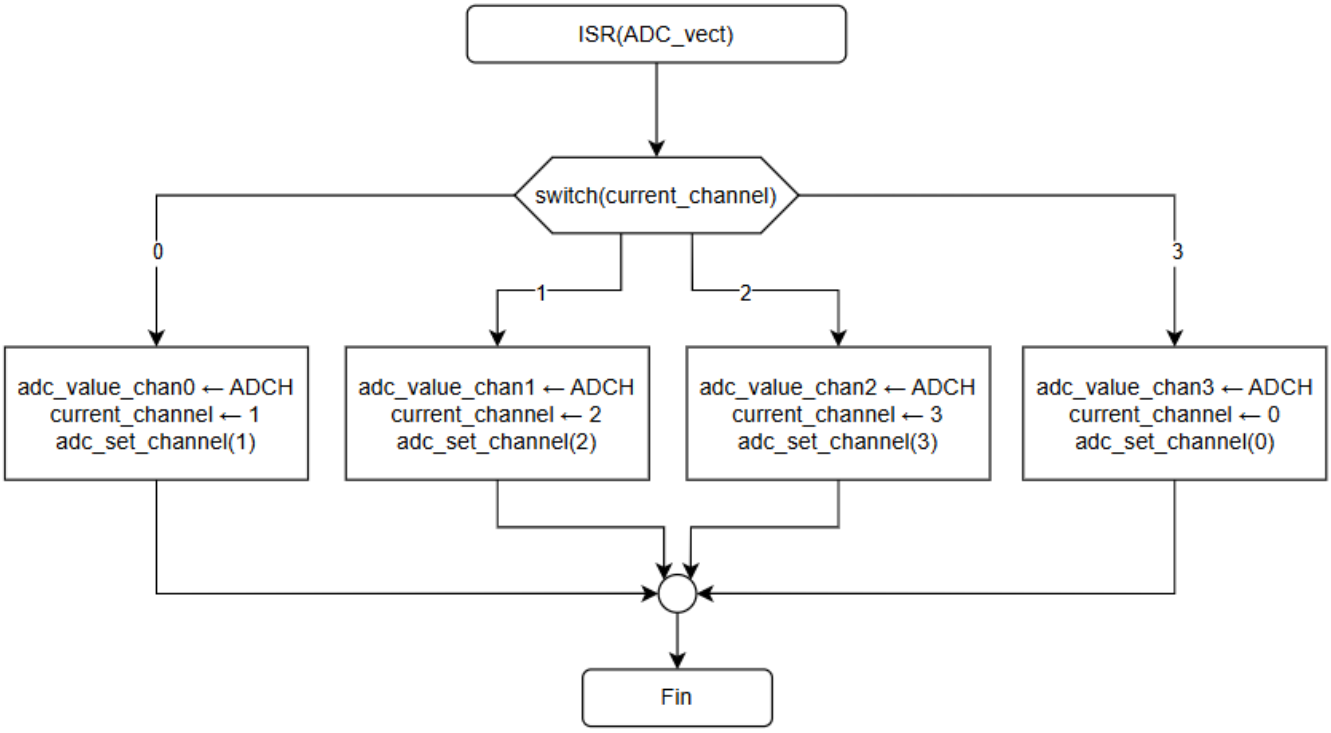

Fig. 3. Multiplexado de canal de ADC en rutina de interrupción de ADC

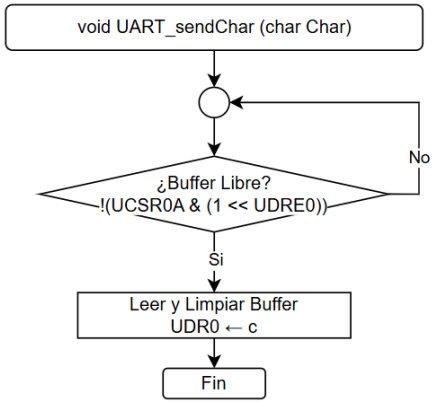

Fig. 4. Enviar caracter con UART

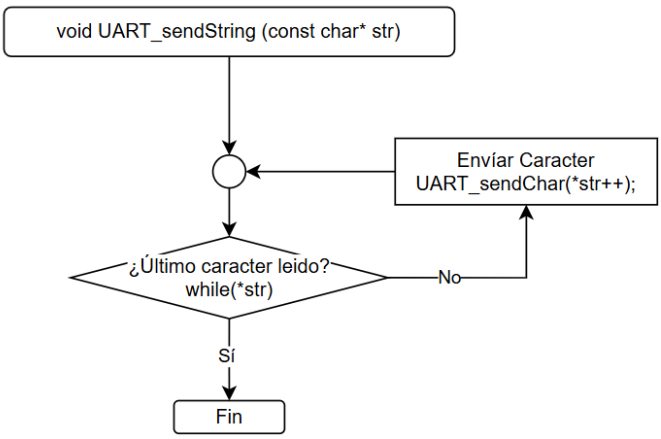

Fig. 5. Enviar cadena de caracteres con UART

una forma simplificada, y las velocidades se saturaron a -255 o 255 de acuerdo con su valor.

Adicionalmente, si las entradas caen en un rango específico denominado *zona muerta*, los motores no giran y los ciclos de trabajo son iguales a cero. De lo contrario, los ciclos de trabajo de los motores se ajustan de acuerdo a una regla de normalización para números de 8 bits. Esta regla hace igual a cero cualquier número próximo a 127, dado que este es el valor medio entre 0 y 255.

D. Servomotores

Se usa el Timer 1 para controlar los servomotores. Como se mencionó anteriormente se utiliza un control proporcional para mover cada motor.

E. Interpretación de Instrucciones

Para interpretar las instrucciones recibidas en el serial, se concatenan los bits de instrucciones recibidos y se introducen en una sentencia switch-case. Si estas son iguales a un par de caracteres ASCII predefinidos, se ejecuta una acción relacionada a un módulo particular. Véase la Figura 12.

F. EEPROM

Para la lectura y escritura en la EEPROM, se utilizaron los códigos proporcionados en el manual del microcontrolador. Estos requieren de que la última escritura haya finalizado para

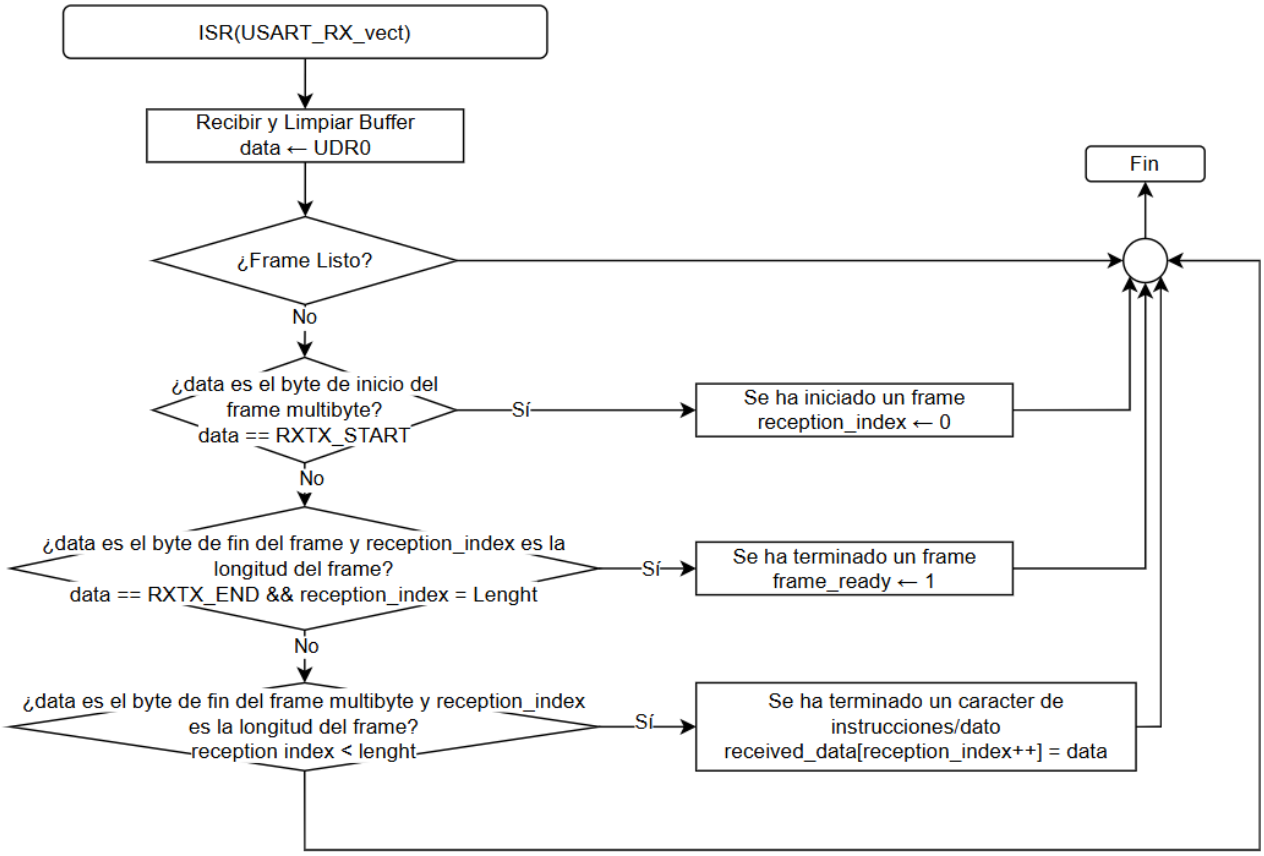

Fig. 6. Interrupción por recepción de caracter en USART.

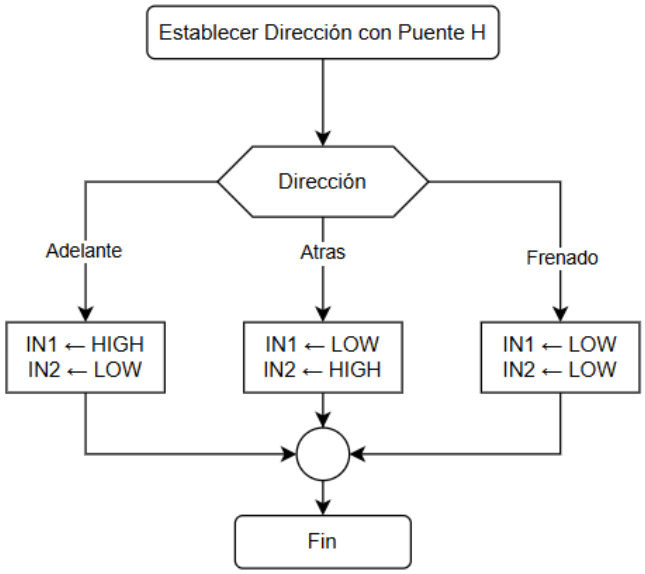

Fig. 7. Control de dirección de movimiento en puente H

poder realizar ya sea una lectura u otra escritura. Véase las figuras 13 y 14.

G. Modo Manual

En el modo manual se utilizan las cuatro entradas del ADC para mover los servomotores y los motorreductores. Esto se facilita con la función *manual_mode_movement* que recopila las funciones previamente vistas.

H. Mainloop

En el ciclo principal se procesan instrucciones de ambos el modo manual y el modo uart, dependiendo del estado del modo manual. Debido a que este es el único estado con una bandera (Ya que el modo UART NO debería deshabilitarse, a menos que haya un botón manual que lo reactive), es necesario apagar el modo manual para que este no haga interferencia con el modo UART. Si el modo UART está activo y hay un frame listo para ser procesado, este se pasa a la función previamente definida para procesar la instrucción y la bandera de frame listo se apaga.

IV. Esquemático

La figura 17 muestra el esquemático del circuito xd.

References

[1] Naimi, Naimi & Mazidi, *The AVR microcontroller and Embedded Systems Using assembly and C.*. Pearson Education. 2011.

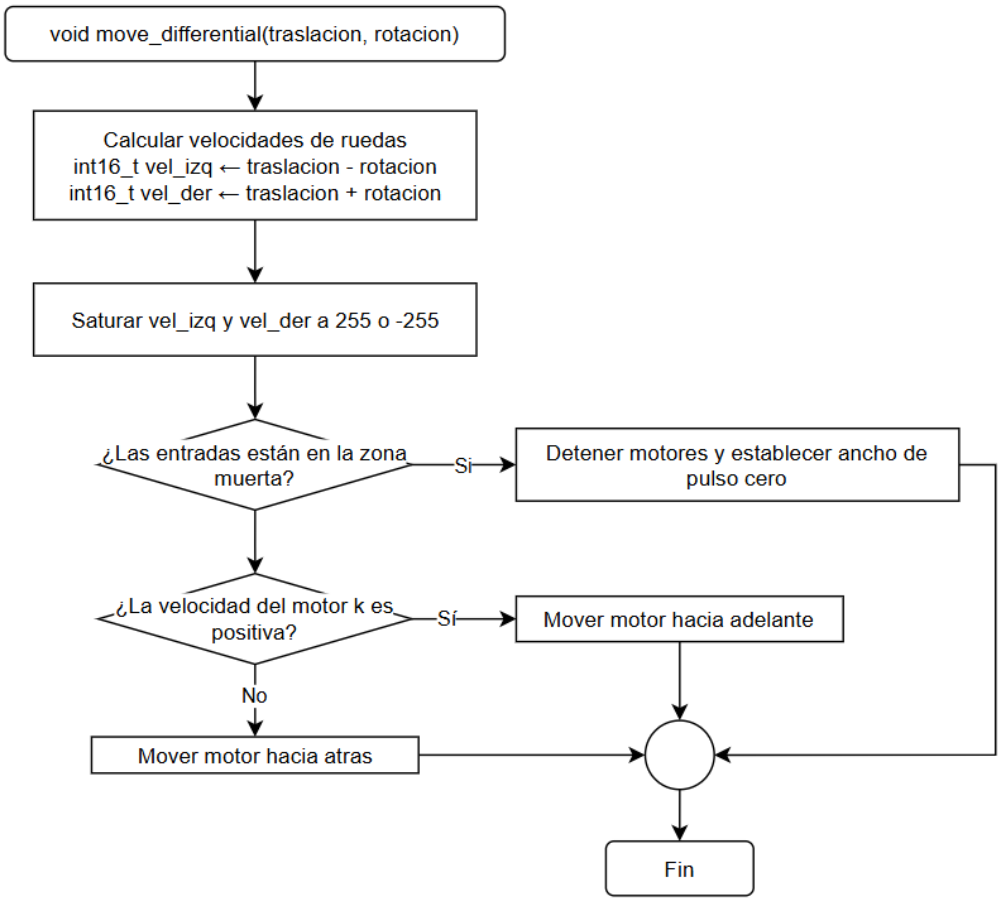

Fig. 8. Control de ruedas diferenciales.

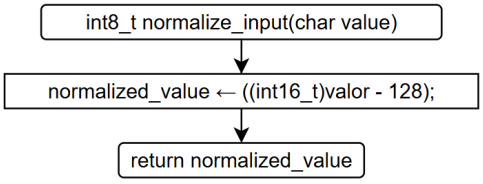

Fig. 9. Normalización de entradas de ADC

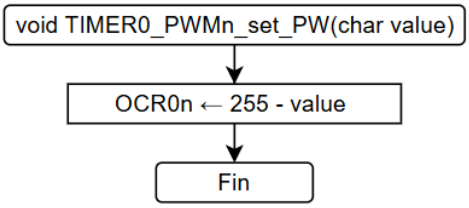

Fig. 10. Control de ciclo de trabajo de Timer0 - Motorreductores

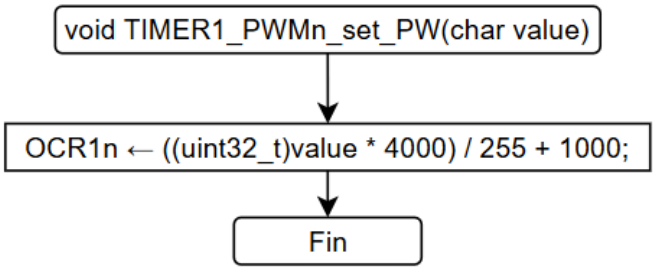

Fig. 11. Control de ciclo de trabajo en Timer1 - Servomotores

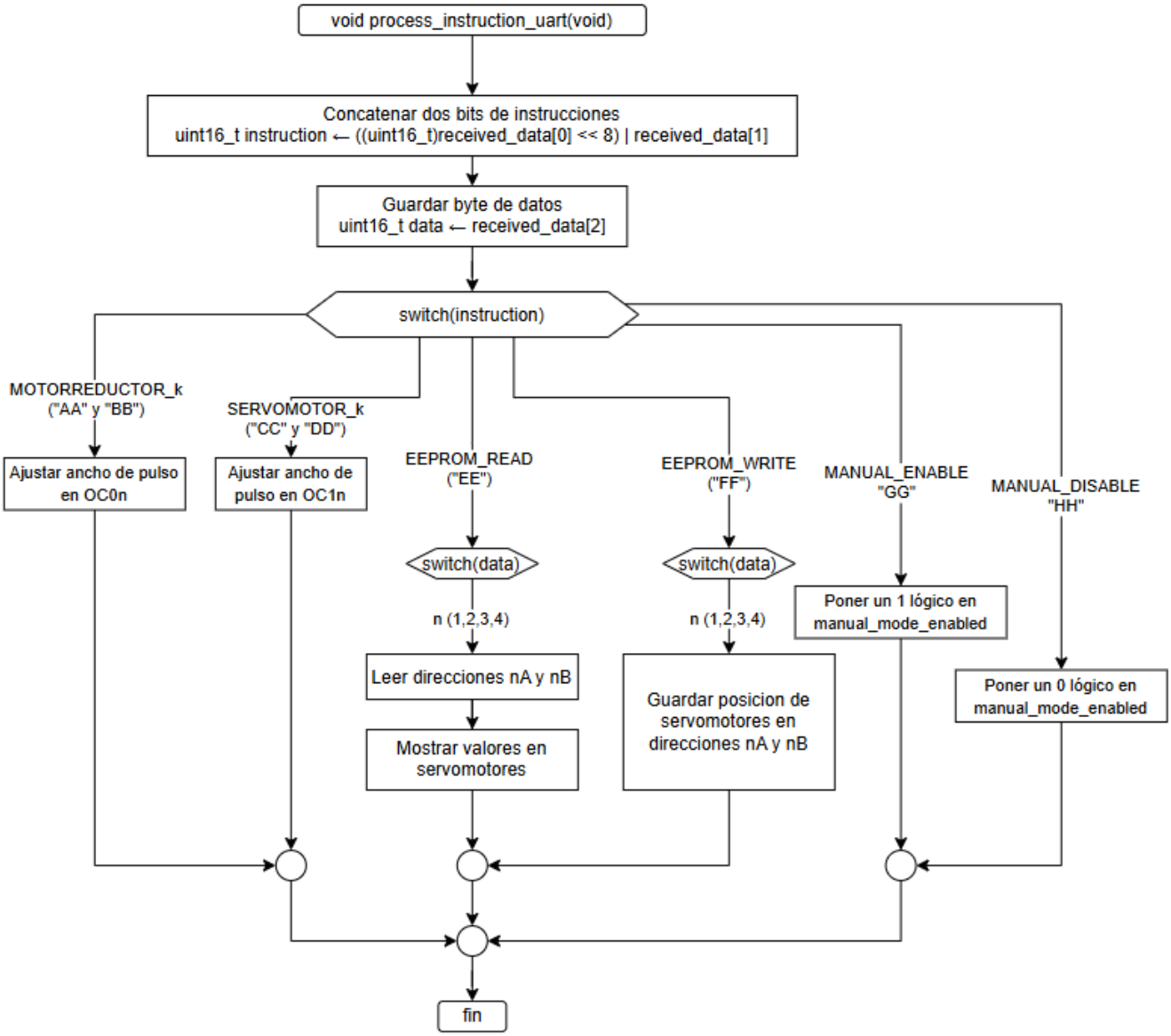

Fig. 12. Procesamiento de Instrucciones

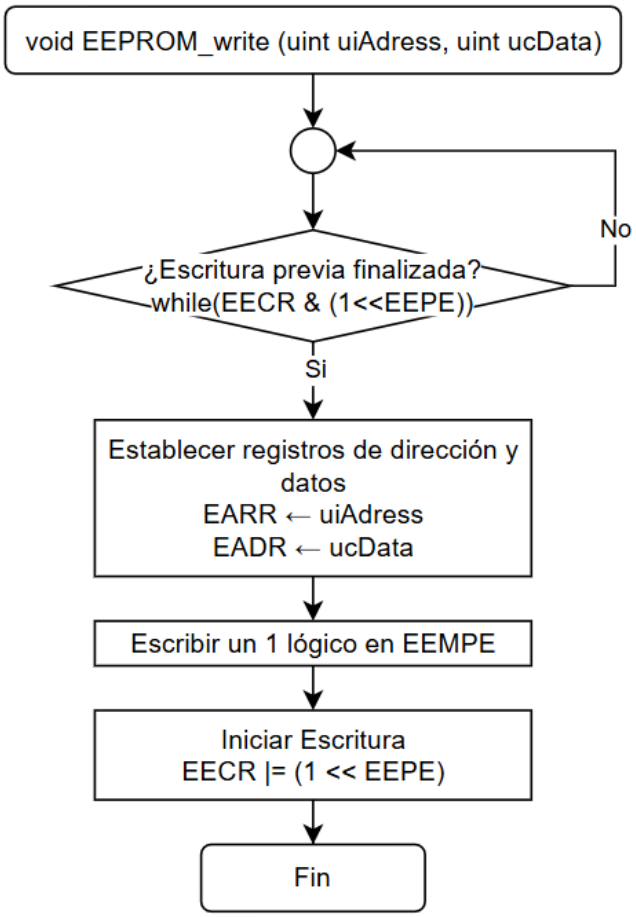

Fig. 13. Escritura en EEPROM

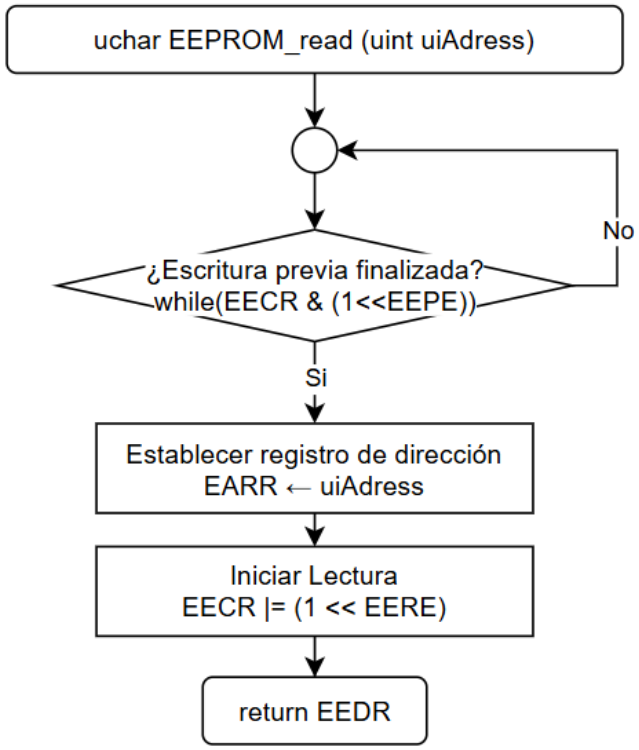

Fig. 14. Enter Caption

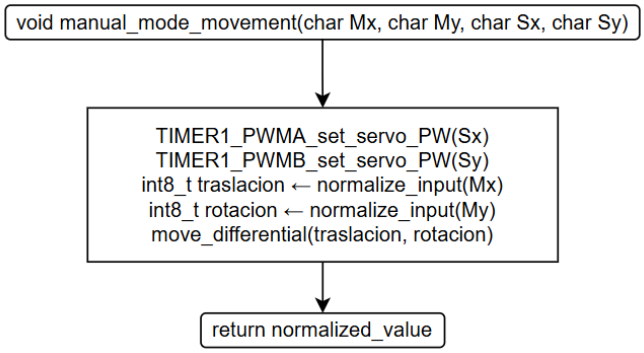

Fig. 15. Modo Manual

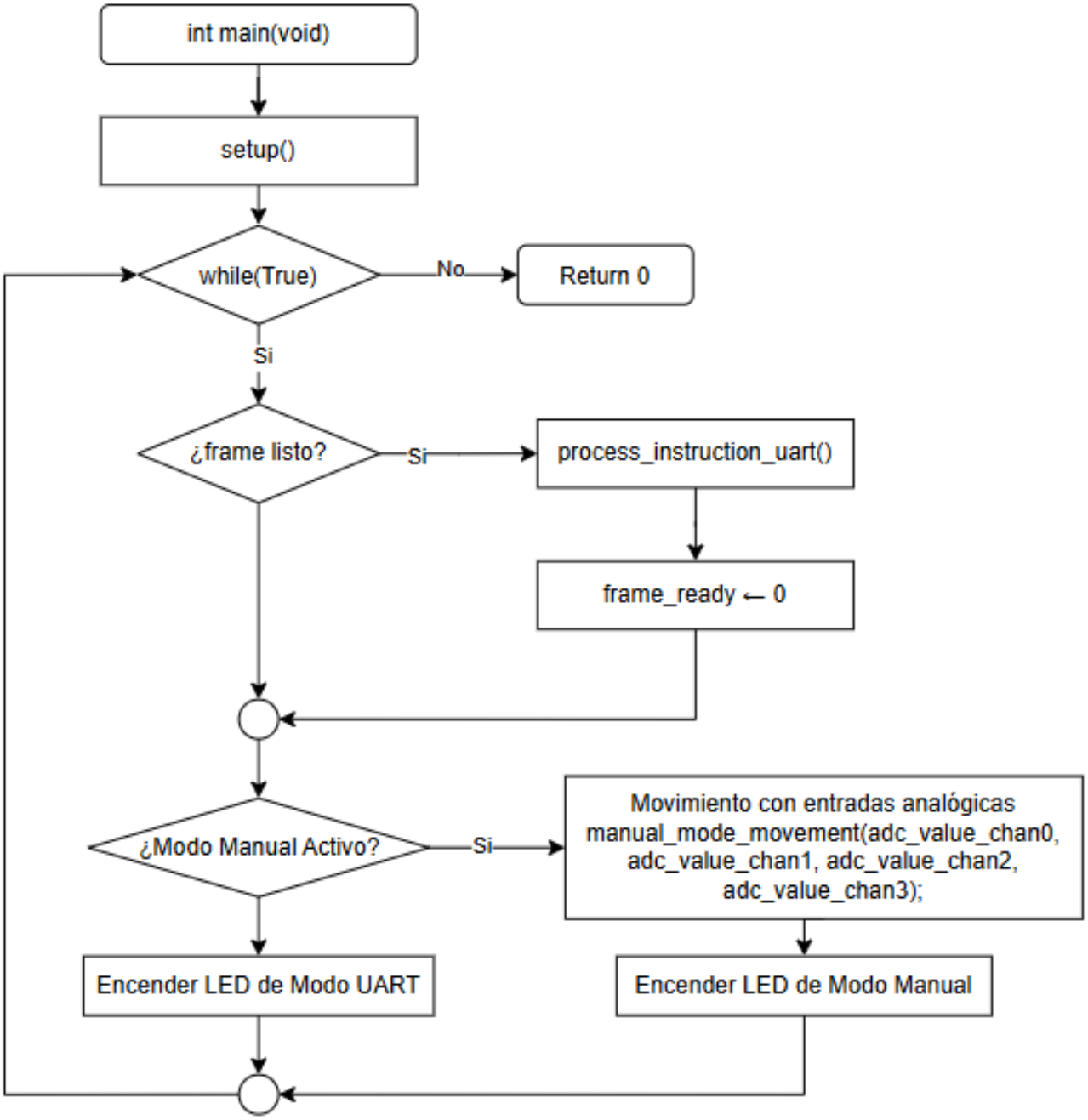

Fig. 16. Mainloop

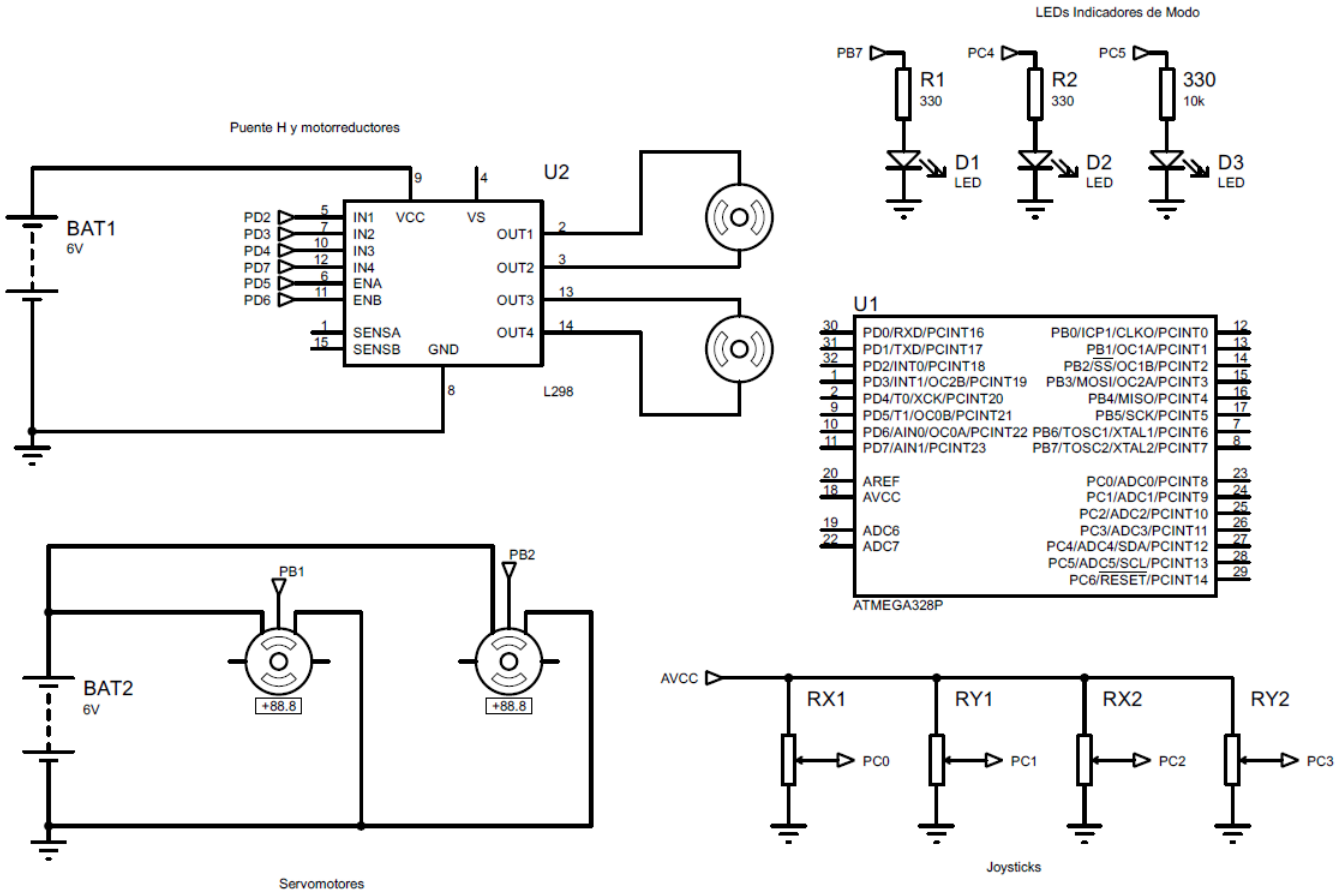

Fig. 17. Esquemático de circuito