(6609) Laboratorio de microprocesadores

Trabajo Práctico 3

Profesor	Ing. Guillermo Campiglio
Cuatrimestre y Año	2C2023
Turno de clases prácticas	Jueves
Jefe de trabajos prácticos	Ratto, Graciela
Docente guía	Gavinowich, Gabriel

Autores			Seguimiento del proyecto							
Nombre	Apellido	Padrón								
Nicolás	Nobili	107540								
Francisco	Russo	107480								

Observaciones:

Fecha de aprobación		

Firma J.T.P

Coloquio	
Nota Final	
Firma profesor	

Índice

1. Objetivo:

El propósito central de este proyecto radica en crear un sensor de distancia con dos grados de libertad de movimiento (Pitch y Yaw) empleando el microcontrolador AtMega328p y el sensor de ultrasonido HC-SR04. Este prototipo se encargará de recibir y enviar comandos a través de una comunicación serial empleando la interfaz Bluetooth, además de interactuar con una página web capaz de interpretar y gestionar los comandos recibidos y transmitidos.

2. Desarrollo:

2.1. Descripción del Proyecto:

El proyecto diseñado utiliza un sensor de ultrasonido HC-SR04 el cual es apuntado utilizando dos servos SG-90. Los datos de medición se procesan en el microcontrolador y son envidados mediante el modulo Bluetooth HC-05 utilizando comunicación serial. Por último, se agrego un láser para señalar la posición apuntada. El microcontrolador usado fue, como se mencionó previamente, un AtMega328p con un cristal externo de 16MHz. En la figura **??** se muestra un diagrama de conexiones en bloques del prototipo:

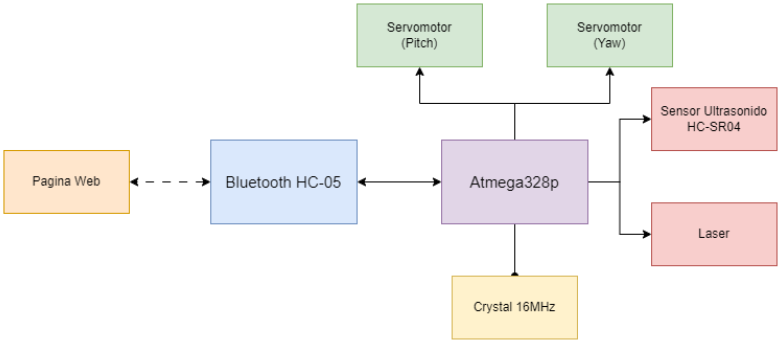

Figura 1: Diagrama de conexión en bloques

El sensor de ultrasonido funciona enviando un disparo por su pin de trigger de un ancho de pulso de 10 μs al menos. Una vez enviado el disparo, el sensor envía una señal de ultrasonido y el pin de ECHO se pone en alto hasta que la señal de audio regreso y va a nivel bajo. En caso de no retorno de la señal de audio, esto se produce automáticamente. Para calcular la distancia medida, se mide el tiempo en alto de la señal de ECHO utilizando el timer 2. Este mismo se configuro con un prescaler de 8, es decir cuenta con frecuencia de 2Mhz. Entonces, según el datasheet:

$$Dist = \frac{t[\mu s]}{2 * 58}$$

Los servos SG90 se controlan mediante dos señales PWM obtenidas de los pines PB1 y PB2 (OCR1A y OCR1B) con el TIMER1. Para obtener dos PWM del TIMER1 se lo configuró en el modo 10 (Phase Correct con TOP = ICR1). Ambas

señales PWM tienen un periodo de 20 ms y los tiempos en alto varían desde 0,5 ms (-90 grados) a 2,5 ms (90 grados). Para el servo encargado del pitch, se restringió su rango de movimiento a entre 0 y 90 grados.

Por otro lado, la comunicación con el módulo Bluetooth HC-05 se hizo con el protocolo USART con un baud rate de 9600. Para la recepción de datos se utilizó la interrupción de recepción de datos del microcontrolador. El Bluetooth es el encargado de conectar el prototipo con la pagina web diseñada para su control. En la sección "Funcionalidades"se describen todos los comandos existentes.

Por último, se implementó que el dispositivo entre en modo sleep cuando no se este ejecutando ningún comando. El mismo permite un uso mas eficiente de la energía y de los componentes. Se agrego un led "active led" que indica si el microcontrolador está dormido o no.

Figura 2: Prototipo final

2.2. Circuito Esquemático:

En la figura 3 se muestra el esquemático del circuito desarrollado en el proyecto.

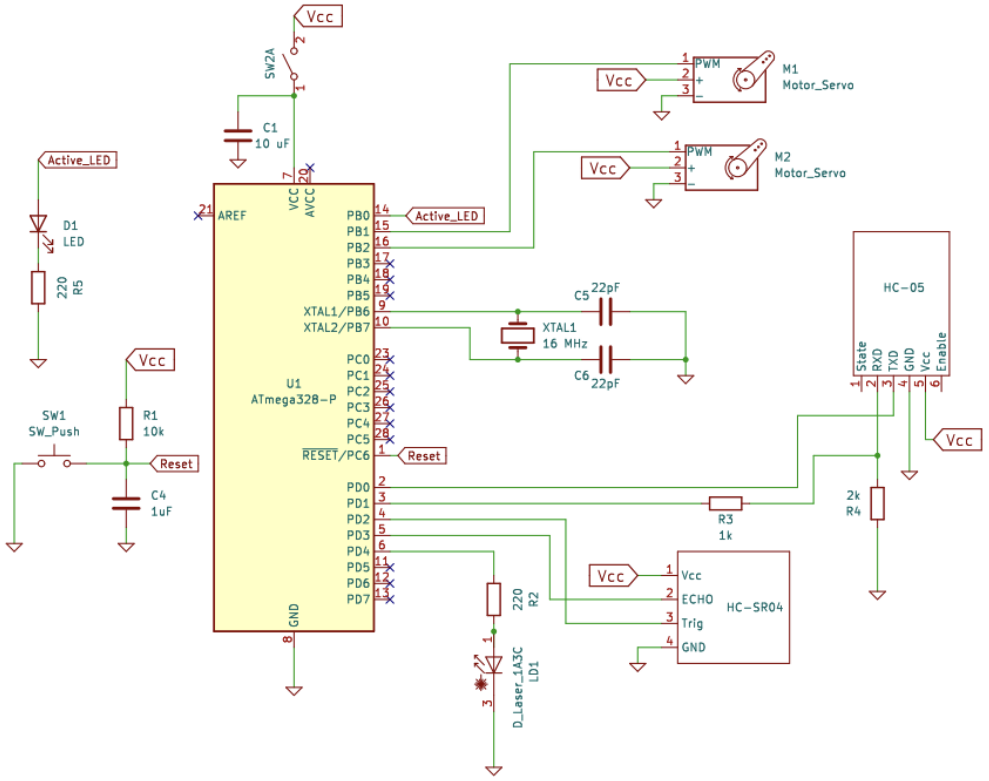

Figura 3: Circuito Esquemático

2.3. Software:

En las figuras 4, 5, 6, 8 y 7 se muestran los diagramas de flujo de la lógica de programación utilizada en el trabajo práctico.

El software se basa principalmente en interrupciones:

- TIMER0: Su interrupción de overflow es responsable de finalizar delays, los cuales pueden ser iniciados por cualquier otra subrutina o interrupción.
- TIMER1: El timer1 es utilizado exclusivamente para la generación de señales PWM.
- TIMER2: Su interrupción de overflow es responsable de mantener el byte alto de una medición. Además, su contador (TCNT2) actúa como byte bajo de la medición.
- PCINT: El único pin habilitado para iniciar interrupciones de Pin Change es el ECHO del sensor de ultrasonido. De esta manera, se puede iniciar y finalizar el TIMER2 y obtener una medición.

- USART: Para la recepción de bytes se utiliza la interrupción de recepción. El procesamiento de dichos bytes se realiza en el main loop. El enviado de bytes se hace de manera bloqueante.

Por otro lado, el flujo principal está encargado de configurar todos los puertos, interrupciones y timers. El loop principal revisa si hay bytes o comandos listos para ser procesados, o si hay que iniciar una medición. De no ser el caso, se ejecuta la instrucción sleep y se apaga el ACTIVE LED.

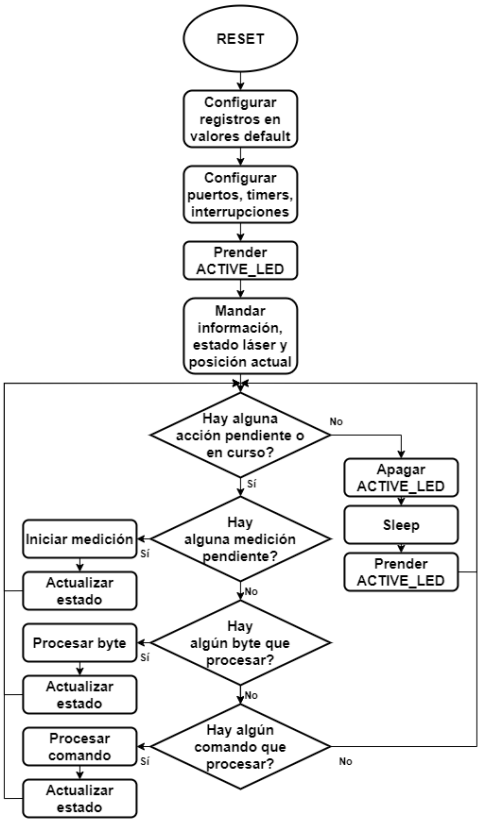

Figura 4: Main

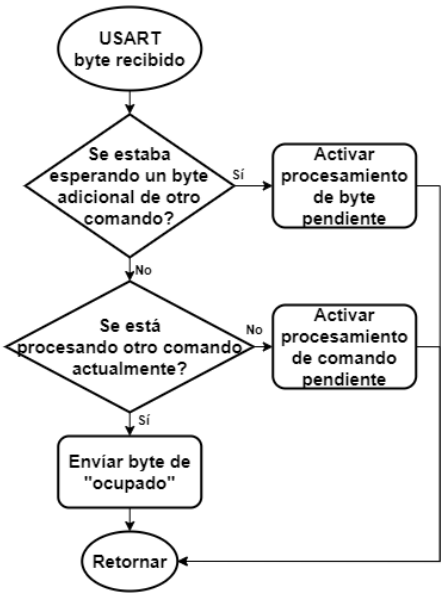

Figura 5: Interrupción de Recepción USART

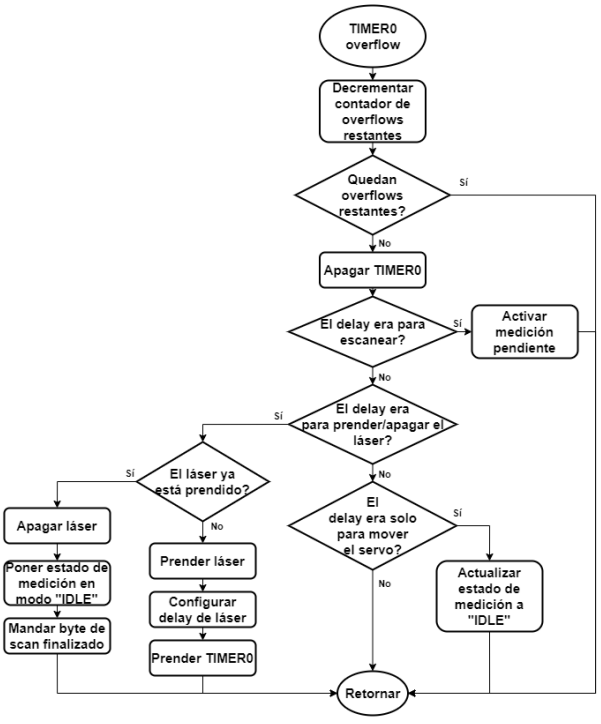

Figura 6: Interrupción por Overflow Timer 0

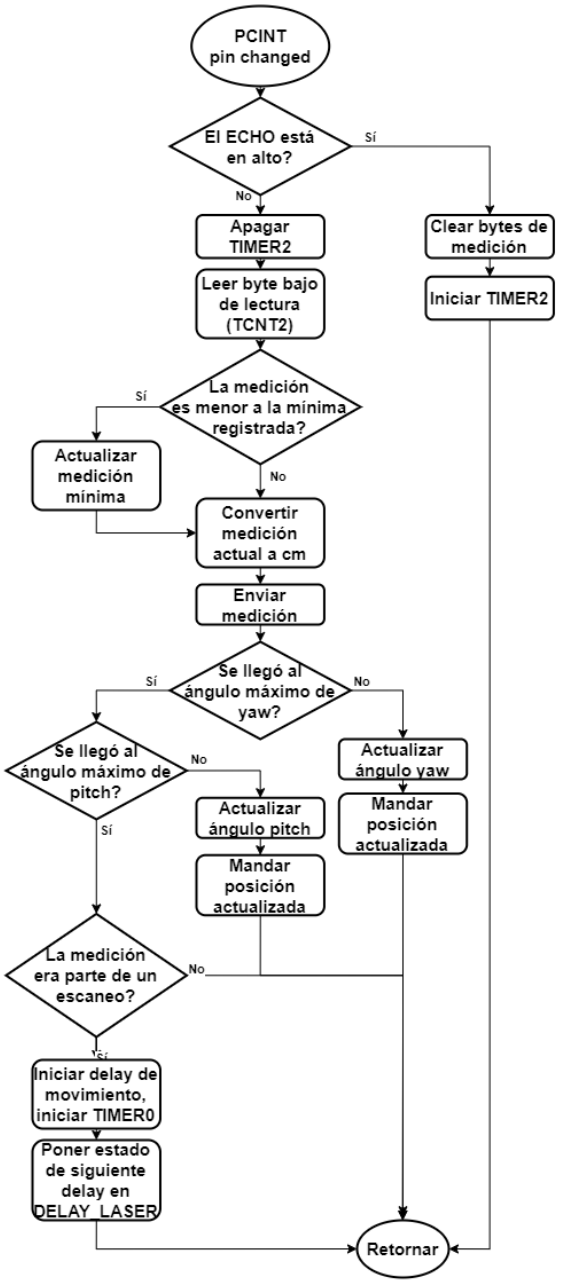

Figura 7: Interrupción por Pin Change del ECHO

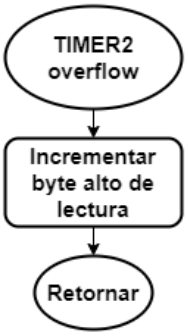

Figura 8: Interrupción por Overflow Timer 2

2.4. Funcionalidades:

A continuación en la tabla 1 se describen la funcionalidades del trabajo y sus comandos correspondientes. Las posiciones que se envian al dispositivo no corresponden a caracteres ASCII.

Nombre	Descripción	Comando
Escanear Fila	Escanea la fila correspondiente a la posición actual y apunta al lugar cuya medición fue la menor	s
Escanear Columna	Ídem a escanear fila pero con la columna correspondiente a la posición actual	t
Escanear Todo	Hace un escaneo completo dado por el rango de movimiento de los servos	z
Escanear Región	Escanea una región rectangular delimitada por (x_i, y_i) y (x_f, y_f)	w $[x_i][y_i]$ $[x_f][y_f]$
Medir Distancia	Mide la distancia en la posición actual	m
Ping	Verifica conexión con el dispositivo	b
Consultar Posición	Consulta la posición actual del prototipo	p
Consultar Láser	Consulta el estado actual del laser	l
Prender Láser	Prende el láser del prototipo	c
Apagar Láser	Apaga el láser del prototipo	d
Mover posición	Mueve el dispositivo a la posición enviada	x $[x_f][y_f]$
Pedir Información	Cuando se envía este comando el dispositivo envía el mensaje almacenado en la EEPROM	i
Escribir Información	Este comando permite modificar el mensaje guardado en la EEPROM	h [texto] NULL

Cuadro 1: Lista de comandos y sus descripciones

En la tabla 2 se muestran las respuestas del dispositivo las cuales envían al ejecutar alguno de los comandos descriptos.

Nombre	Descripción	Respuesta
Escaneo finalizado	Indicación de que se realizó el escaneo completo, incluido el prendido y apagado del láser	f
Medición	La medición en la posición (x, y) fue de $high$:low centímetros	m [x][y] [low][$high$]
Posición actual	La posición de los servos actual es (x, y)	p [x][y]
Láser prendido	Indicación de que el láser está prendido	j
Láser apagado	Indicación de que el láser está apagado	k
Pong	Respuesta exitosa al comando Ping	b
Ocupado	No se puede procesar el comando porque se está realizando un escaneo	n
Comando desconocido	El último comando recibido no es ningún comando válido	w
Escritura completa	La sobre-escritura del mensaje guardado en EEPROM finalizó	h
Mensaje en EEPROM	El mensaje guardado en EEPROM es "texto"	i [texto] NULL

Cuadro 2: Lista de respuestas y sus descripciones

2.5. Modelos 3D:

A continuación se adjuntan capturas del modelo 3D diseñado para el prototipo. El mismo fue realizado utilizando el software FreeCad.

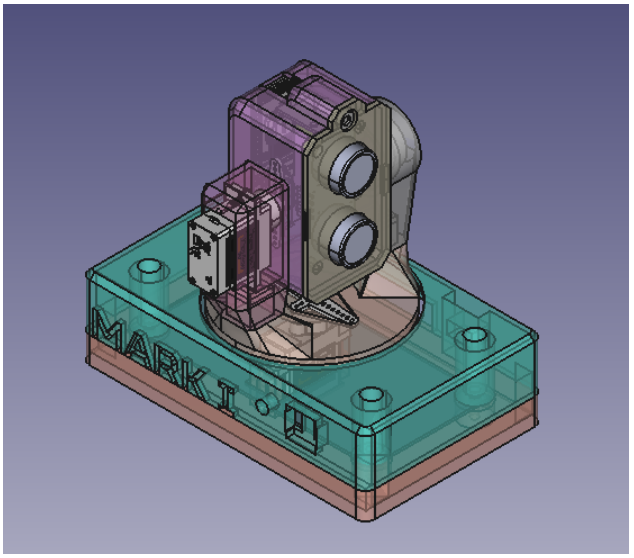

Figura 9: Vista Superior

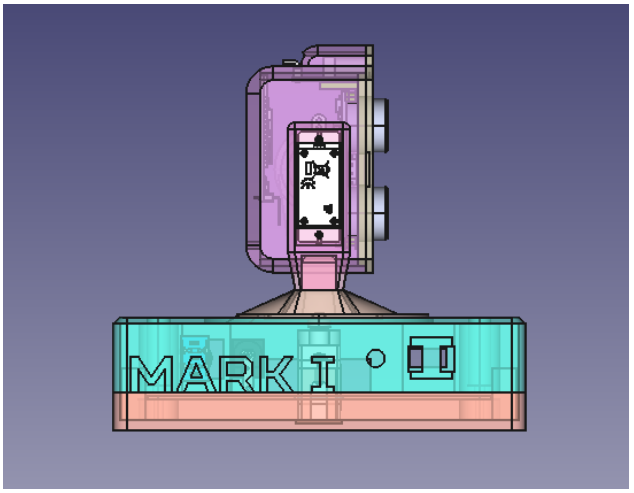

Figura 10: Vista Frontal

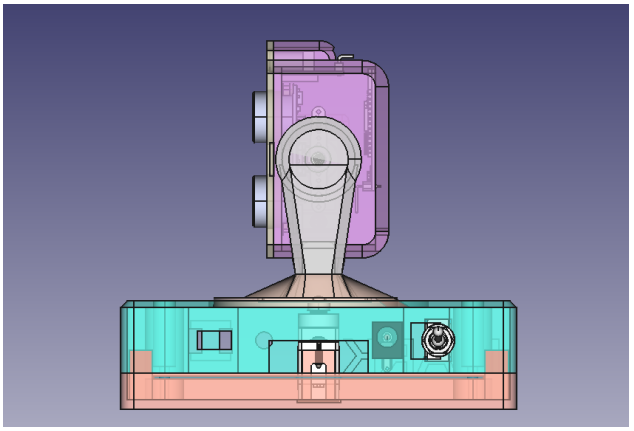

Figura 11: Vista Trasera

2.6. Inferfaz web

Se diseñó una página web utilizando SvelteKit para comunicarse con el prototipo. La comunicación serial a través de Bluetooth fue facilitada gracias al Web Serial API, una interfaz experimental que actualmente está disponible para navegadores basados en Chromium, en PC únicamente.

La grilla central, además de ser interactiva, es una representación visual de las mediciones, cuyos valores numéricos están presentes en la sección inferior derecha. Además, se presentan botones que facilitan la transmisión de comandos, abstrayendo al usuario de los bytes específicos que se requieren enviar. Finalmente, hay un historial de bytes recibidos y enviados, presentados en ASCII y en hexadecimal.

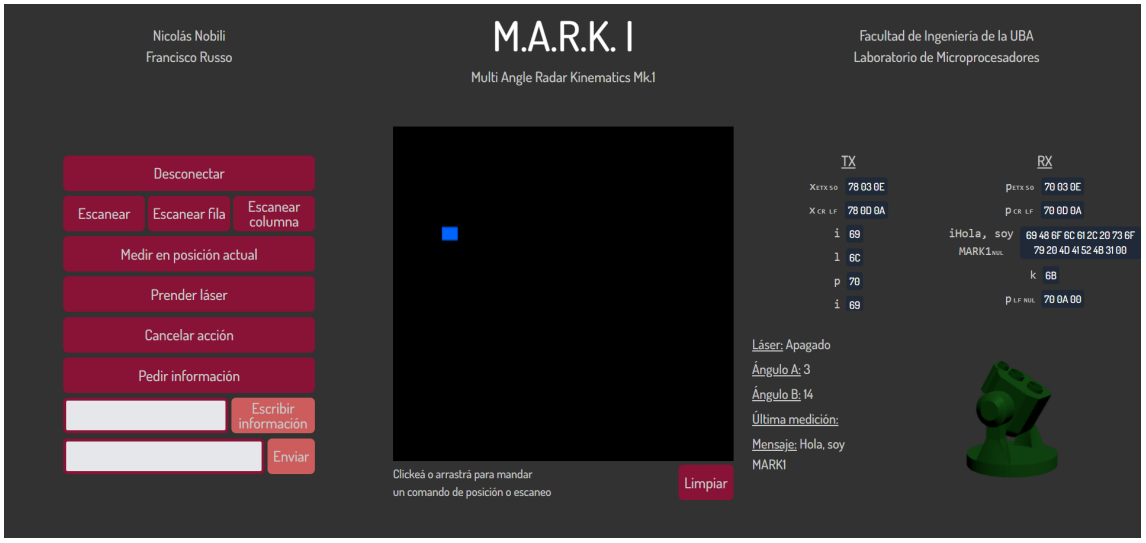

Figura 12: Interfaz web: nicolasnobili.github.io/MARK-1

2.7. Lista de componentes:

Componente	Cantidad	Precio x Unidad
Resistencia 10kΩ	1	$ 16,0
Resistencia 1kΩ	1	$ 16,0
Rulemán 608 RS	1	$ 1000,0
Pulsador	1	$ 400,0
Servo SG90	2	$ 2600,0
AtMega328p	1	$ 3923,5
Cristal 16 MHz	1	$ 245,0
Capacitor 22 pF	2	$ 49,0
Capacitor 10 μF	1	$ 40,0
Capacitor 100 nF	1	$ 40,0
HC-05 (Bluetooth)	1	$ 6031,5
HC-SR04 (Sensor Ultrasonido)	1	$ 2009,7
Placa Experimental 100x50	1	$ 2412,3
Laser Module HW-493	1	$ 1510,0
DC Jack 2.1 mm	1	$ 367,0
DC Plug 2.1 mm	1	$ 458,0
	Total:	$ 23791,6

Cuadro 3: Lista de precios y componentes

2.8. Resultados:

El comportamiento del prototipo fue el esperado. Se probaron todas sus funcionalidades y todas funcionan correctamente. Al hacer alguno de los barridos, área, fila o columna; el sensor apunta a aquella posición en la que se midió la menor distancia y enciende el puntero láser.

En cuanto al consumo de energía, se debe destacar que se midió al realizar pruebas con el prototipo real que puede consumir hasta picos de 2 A cuando los servos realizan movimiento largos. Lamentablemente no se encontró información sobre el consumo de corriente de los servos en sus datasheets. La tensión de funcionamiento del prototipo es de 5 V y consume 30 mA en reposo.

3. Conclusión:

Se cumplieron los objetivos propuestos en cuanto al funcionamiento teórico del prototipo y la medición de distancias con el sensor ultrasonido. Además, se pudo integrar de forma exitosa la comunicación serial con Bluetooth entre el prototipo y su interfaz gráfica diseñada con Javascript.

En cuanto a errores o comportamientos no deseados, se observa al probar el prototipo e intentar que apunte a cierto objeto que este generalmente apunta a un lado del mismo. Esto se debe al gran ángulo de medición que tiene el sensor utilizado dado por la naturaleza de la propagación de ondas de sonido. El sensor al

medir la presencia de un objeto, apunta a la primer posición de menor distancia, por lo tanto si el objeto es detectado varias veces, el sensor apuntara a la primer posición en donde lo detecto. Esto puede solucionarse fácilmente utilizando un sensor cuyo ángulo de dispersión sea menor, como por ejemplo un sensor óptico infrarrojo que de mediciones mas precisas. Por otro lado, para combatir el ruido de las mediciones se podría medir varias veces en rápida sucesión y luego promediar las muestras.

El proyecto podría mejorarse cambiando el sistema de comunicación implementado. En su estado presente, no hay robustez en las comunicaciones en el caso de una transmisión rápida de bytes, uno tras el otro. La mejora puede hacerse implementando un buffer de recepción y utilizando un carácter centinela para indicar el fin de los mensajes. Esto también permitiría mayor flexibilidad a la hora de agregar o cambiar funcionalidades del proyecto. Además, se podría implementar un buffer adicional para la transmisión de bytes en conjunto con interrupciones de transmisión.

El movimiento de los servos tiene espacio para mejoras: En primer lugar, su movimiento podría ser suavizado si se desarrollara un software con control más granular y continuo de las señales PWM. Esto evitaría movimientos repentinos y bruscos que pueden llegar a dañar o desgastar al prototipo. Por otro lado, al iniciar escaneos, se podría buscar la esquina del rectángulo más cercana a la posición actual para evitar movimientos innecesarios.

4. Apéndice:

```
; ---------------------------------
; MARK1_include.asm
;
; Created: 11/11/2023 11:40:49 AM
; Authors: FR & NN
; ---------------------------------


; -----------------------------------------------------
;                 CONSTANTES AUXILIARES
; -----------------------------------------------------

.equ CLK_FREQUENCY = 16000000

.equ PWM_PERIOD_US  = 20000
.equ PWM_MAX_TON_US_A = 2500
.equ PWM_MIN_TON_US_A = 500
.equ PWM_MAX_TON_US_B = 1700
.equ PWM_MIN_TON_US_B = 500

.equ BAUD_RATE = 9600


; -----------------------------------------------------
;                  CONSTANTES NUM RICAS
; -----------------------------------------------------

.equ MAX_STEPA = 20  ; Hay MAX_STEPA estados  m  s  el 0 = MAX_STEPA
   + 1 estados
.equ MAX_STEPB = 20  ; Hay MAX_STEPB estados  m  s  el 0 = MAX_STEPB
   + 1 estados
.equ STEPA_INICIAL = 10
.equ STEPB_INICIAL = 0

.equ TOP_PWM = int( (CLK_FREQUENCY/8) * PWM_PERIOD_US / (2 *
   1000000) ) ; del datasheet

.equ MAX_OCR1A  = int(TOP_PWM - TOP_PWM * PWM_MIN_TON_US_A /
   PWM_PERIOD_US)
.equ MIN_OCR1A  = int(TOP_PWM - TOP_PWM * PWM_MAX_TON_US_A /
   PWM_PERIOD_US)
.equ STEP_OCR1A = (MAX_OCR1A - MIN_OCR1A) / MAX_STEPA

.equ MAX_OCR1B  = int(TOP_PWM - TOP_PWM * PWM_MIN_TON_US_B /
   PWM_PERIOD_US)
.equ MIN_OCR1B  = int(TOP_PWM - TOP_PWM * PWM_MAX_TON_US_B /
   PWM_PERIOD_US)
.equ STEP_OCR1B = (MAX_OCR1B - MIN_OCR1B) / MAX_STEPB ; si llegaste
    ac  , eleg  valores que den todo entero bro

.equ UBRR0 = int( CLK_FREQUENCY / (16 * BAUD_RATE) - 1 )

; Medidos en overflows del timer 0 (16 ms)
```

```
.equ DELAY_MOVIMIENTO     = 20
.equ DELAY_STEP           = 10
.equ DELAY_SINGLE_MEASURE = 2
.equ DELAY_DURACION_LASER = 0xAF

; Objetivo: 10 us
.equ LOOPS_TRIGGER = 55

.equ MAX_STRING = 0xFF

.equ DIVISOR_CONVERSION = 58 * 2

; -----------------------------------------------------
;                 DIRECCIONES EEPROM
; -----------------------------------------------------

.equ INFO_ADDR = 0x0000


; -----------------------------------------------------
;                   PINES Y PUERTOS
; -----------------------------------------------------

; PORTB:
.equ SERVOA_PIN      = 1
.equ SERVOB_PIN      = 2
.equ ACTIVE_LED      = 0

; PORTD:
.equ BT_RX     = 0
.equ BT_TX     = 1
.equ ULTRASOUND_TRIG = 2
.equ ULTRASOUND_ECHO = 3
; .equ INT0_PIN  = 2
.equ LASER_PIN = 4

;PORTC:
.equ RESET_PIN = 6


; -----------------------------------------------------
;                 ESTADOS Y OBJETIVOS
; -----------------------------------------------------

; ESTADOS DE MEDICI N:
.equ WAIT_MEDIR       = 0x00
.equ MEDIR            = 0x01
.equ MIDIENDO         = 0x02
.equ DELAY_SCAN       = 0x03
.equ DELAY_MOVE_TO    = 0x04
.equ DELAY_LASER      = 0x05

; ESTADOS DE COMUNICACI N:
.equ WAIT_COMMAND     = 0x00
.equ PROCESAR_COMANDO = 0x01
```

```
.equ WAIT_BYTE         = 0x02
.equ PROCESAR_BYTE     = 0x03


; -------------------------------------------------------
;                       COMUNICACI N
; -------------------------------------------------------

; COMANDOS:
.equ ABORT           = 'a'
.equ SCAN_ROW        = 's'
.equ SCAN_COL        = 't'
.equ SCAN_ALL        = 'z'
.equ SCAN_REGION     = 'w'
.equ MEDIR_DIST      = 'm'
.equ PING            = 'b'
.equ ASK_POSITION    = 'p'
.equ ASK_LASER       = 'l'
.equ TURN_ON_LASER   = 'c'
.equ TURN_OFF_LASER  = 'd'
.equ MOVE_TO         = 'x'
.equ WRITE_INFO      = 'h'
.equ ASK_INFO        = 'i'
.equ ASK_STATE       = 'y'

; DATA TYPES:
.equ SCAN_DONE         = 'f'
.equ MEASUREMENT       = 'm'
.equ CURRENT_POSITION  = 'p'
.equ LASER_ON          = 'j'
.equ LASER_OFF         = 'k'
.equ PONG              = 'b'
.equ DEBUG             = 'o'
.equ BUSY              = 'n'
.equ WHAT              = 'w'
.equ WRITE_INFO_DONE   = 'h'
.equ INFO              = 'i'
.equ MEASURE_STATE     = 'y'


; -------------------------------------------------------
;                         REGISTROS
; -------------------------------------------------------

.def zero               = r0


.def loop_index         = r3
.def first_stepa        = r4
.def first_stepb        = r5
.def last_stepa         = r6
.def last_stepb         = r7
.def bytes_restantes    = r8
.def lectural           = r9
.def lecturah           = r10
```

```
.def min_distl          = r11
.def min_disth          = r12
.def min_stepa          = r13
.def min_stepb          = r14

.def temp               = r16
.def stepa              = r17
.def stepb              = r18
.def estado_comando     = r19
.def estado_medicion    = r20
.def comando_recibido   = r21
.def left_ovfs          = r22
.def data_type          = r23
.def temp_byte          = r24
.def byte_recibido      = r25



; ---------------------------------------------------------
;                           MACROS
; ---------------------------------------------------------

.macro ldx
  ldi XL, LOW(@0)
  ldi XH, HIGH(@0)
.endmacro

.macro ldy
  ldi YL, LOW(@0)
  ldi YH, HIGH(@0)
.endmacro

.macro ldz
  ldi ZL, LOW(@0)
  ldi ZH, HIGH(@0)
.endmacro

; ***** END OF FILE
   ******************************************************
```

```
; ---------------------------------
; main.asm
;
; Created: 11/11/2023 10:50:07 AM
; Authors: FR & NN
; ---------------------------------

.include "MARK1_include.asm"


; ---------------------------------------------------------
;                     MEMORIA EEPROM
; ---------------------------------------------------------
.eseg
.org INFO_ADDR

```

```
default_info:
    .db "Connected to MARK1: Multi Angle Radar Kinematics Mk.1", 0


; -----------------------------------------------------
;                    MEMORIA DE DATOS
; -----------------------------------------------------

.dseg
.org SRAM_START

buffer: .byte MAX_STRING
lectura_ascii: .byte 4  ; El m ximo es FFFF


; -----------------------------------------------------
;                VECTOR DE INTERRUPCIONES
; -----------------------------------------------------

.cseg
.org 0x0000
    rjmp main
.org INT0addr
    rjmp handler_INT0
.org PCI2addr
  rjmp handler_PCI2
.org URXCaddr
    rjmp handler_URXC
.org OVF0addr
    rjmp handler_OVF0
.org OVF2addr
    rjmp handler_OVF2


; -----------------------------------------------------
;                  INICIALIZACION
; -----------------------------------------------------

.org INT_VECTORS_SIZE
main:
  ; Stack pointer
  ldi temp, LOW(RAMEND)
  out spl, temp
  ldi temp, HIGH(RAMEND)
  out sph, temp

  clr zero
    ldi stepa, STEPA_INICIAL
    ldi stepb, STEPB_INICIAL
    ldi estado_medicion, WAIT_MEDIR
    ldi estado_comando, WAIT_COMMAND

  rcall config_ports
  rcall config_timer0
  rcall config_timer1
```

```
72    rcall config_timer2
73    rcall config_USART
74    ; rcall config_int0
75      rcall config_pci0
76
77    sbi PORTB, ACTIVE_LED
78
79    ldi data_type, CURRENT_POSITION
80    rcall send_data
81
82    sbis PORTD, LASER_PIN
83      ldi data_type, LASER_OFF
84      sbic PORTD, LASER_PIN
85      ldi data_type, LASER_ON
86      rcall send_data
87
88    ldi data_type, INFO
89    rcall send_data
90
91    sei
92
93
94  ; ------------------------------------------------------
95  ;                   LOOP PRINCIPAL
96  ; ------------------------------------------------------
97
98  main_loop:
99
100     ; Solo dormir si no hay ni comandos
101     ; ni mediciones pendientes
102   cpi estado_medicion, WAIT_MEDIR
103   brne main_loop_accion_pendiente
104     cpi estado_comando, WAIT_COMMAND
105     breq main_sleep
106
107 main_loop_accion_pendiente:
108
109     cpi estado_medicion, MEDIR
110     breq main_iniciar_medicion
111
112     cpi estado_comando, PROCESAR_BYTE
113     breq main_procesar_byte
114
115     cpi estado_comando, PROCESAR_COMANDO
116     breq main_procesar_comando
117
118   rjmp main_loop
119
120
121 ; ------------------------------------------------------
122 ;            SLEEP
123 ; ------------------------------------------------------
124
125
126 main_sleep:
```

```
127     cbi PORTB, ACTIVE_LED
128
129     ; Modo Idle. Mantiene prendida la USART para despertarse
130   ldi temp, (0 << SM2) | (0 << SM1) | (0 << SM0) | (1 << SE)
131     out SMCR, temp
132   sleep
133   out SMCR, zero
134
135     sbi PORTB, ACTIVE_LED
136   rjmp main_loop
137
138
139 ; -----------------------------------------------------
140 ;                   INICIAR MEDICION
141 ; -----------------------------------------------------
142
143 main_iniciar_medicion:
144     ; Esperamos que el ECHO haga una interrupci n
145     ; Mientras tanto no hay que hacer nada
146     ldi estado_medicion, MIDIENDO
147     rcall send_trigger
148
149     rjmp main_loop
150
151
152 ; -----------------------------------------------------
153 ;                   PROCESAR BYTE
154 ; -----------------------------------------------------
155
156 main_procesar_byte:
157     ; Vemos para qu  quer amos este byte
158     cpi comando_recibido, MOVE_TO
159     breq comando_byte_move_to
160
161     cpi comando_recibido, SCAN_REGION
162     breq comando_byte_scan_region
163
164     cpi comando_recibido, WRITE_INFO
165     breq comando_byte_write_info
166
167     ; No deber amos llegar ac
168     rjmp main_loop
169
170 comando_byte_move_to:
171     rcall rutina_comando_byte_move_to
172     rjmp main_loop
173
174 comando_byte_scan_region:
175     rcall rutina_comando_byte_scan_region
176     rjmp main_loop
177
178 comando_byte_write_info:
179     rcall rutina_comando_byte_write_info
180     rjmp main_loop
181
```

```
182
183 ; -------------------------------------------------------
184 ;                  PROCESAR COMANDO
185 ; -------------------------------------------------------
186
187 main_procesar_comando:
188     ; La lectura de los siguientes comandos no modifican el estado
189     ; de la medici n (a excepci n de ABORT) y se pueden realizar
    siempre
190
191     cpi comando_recibido, ABORT
192     breq comando_abort
193
194     cpi comando_recibido, PING
195     breq comando_ping
196
197     cpi comando_recibido, ASK_POSITION
198     breq comando_ask_position
199
200     cpi comando_recibido, ASK_LASER
201     breq comando_ask_laser
202
203   cpi comando_recibido, ASK_STATE
204   breq comando_ask_state
205
206   cpi comando_recibido, ASK_INFO
207   breq comando_ask_info
208
209     ; Para otros comandos, primero verificamos
210     ; Si no hay alguna medici n en curso
211     cpi estado_medicion, WAIT_MEDIR
212     brne send_busy
213
214     ; Leer comando
215
216     cpi comando_recibido, SCAN_ROW
217     breq comando_scan_row
218
219   cpi comando_recibido, SCAN_COL
220     breq comando_scan_col
221
222   cpi comando_recibido, SCAN_ALL
223     breq comando_scan_all
224
225   cpi comando_recibido, SCAN_REGION
226     breq comando_scan_region
227
228     cpi comando_recibido, MOVE_TO
229     breq comando_move_to
230
231     cpi comando_recibido, MEDIR_DIST
232     breq comando_medir_dist
233
234     cpi comando_recibido, TURN_ON_LASER
235     breq comando_turn_on_laser
```

```
236
237     cpi comando_recibido, TURN_OFF_LASER
238     breq comando_turn_off_laser
239
240     cpi comando_recibido, WRITE_INFO
241     breq comando_write_info
242
243     ; Comando desconocido
244     ldi data_type, WHAT
245     rcall send_data
246   ldi estado_comando, WAIT_COMMAND
247     rjmp main_loop
248
249
250 send_busy:
251     ldi data_type, BUSY
252     rcall send_data
253   ldi estado_comando, WAIT_COMMAND
254
255     rjmp main_loop
256
257 comando_abort:
258     rcall rutina_comando_abort
259     rjmp main_loop
260
261 comando_ping:
262   rcall rutina_comando_ping
263     rjmp main_loop
264
265 comando_ask_position:
266   rcall rutina_comando_ask_position
267     rjmp main_loop
268
269 comando_ask_laser:
270   rcall rutina_comando_ask_laser
271     rjmp main_loop
272
273 comando_ask_state:
274   rcall rutina_comando_ask_state
275   rjmp main_loop
276
277 comando_ask_info:
278   rcall rutina_comando_ask_info
279   rjmp main_loop
280
281 comando_scan_row:
282   rcall rutina_comando_scan_row
283   rjmp main_loop
284
285 comando_scan_col:
286   rcall rutina_comando_scan_col
287   rjmp main_loop
288
289 comando_scan_all:
290   rcall rutina_comando_scan_all
```

```
  rjmp main_loop

comando_scan_region:
  rcall rutina_comando_scan_region
  rjmp main_loop

comando_move_to:
  rcall rutina_comando_move_to
    rjmp main_loop

comando_medir_dist:
  rcall rutina_comando_medir_dist
    rjmp main_loop

comando_turn_on_laser:
  rcall rutina_comando_turn_on_laser
    rjmp main_loop

comando_turn_off_laser:
  rcall rutina_comando_turn_off_laser
    rjmp main_loop

comando_write_info:
    rcall rutina_comando_write_info
    rjmp main_loop

end_main:
    rjmp end_main


; ----------------------------------------------------
;                 RUTINAS Y HANDLERS
; ----------------------------------------------------

.include "MARK1_config.asm"
.include "MARK1_handlers.asm"
.include "MARK1_rutina.asm"
```

```
    ; ---------------------------------
; MARK1_config.asm
;
; Created: 11/11/2023 11:49:48 AM
; Authors: FR & NN
; ---------------------------------


; ----------------------------------------------------
;                       PUERTOS
; ----------------------------------------------------

config_ports:
    ; Evitar pulso no deseado al comienzo del programa
  out PORTB, zero

  ; PORTB
  sbi DDRB,  SERVOA_PIN
```

```
19      sbi DDRB,  SERVOB_PIN
20      sbi DDRB,  ACTIVE_LED
21
22    ; PORTD
23    sbi DDRD,  LASER_PIN
24      ; cbi DDRD,  INT0_PIN
25      ; sbi PORTD, INT0_PIN
26    cbi PORTD, ULTRASOUND_TRIG
27    sbi DDRD,  ULTRASOUND_TRIG
28    cbi DDRD,  ULTRASOUND_ECHO
29
30    ; PORTC
31    sbi DDRC,  RESET_PIN
32    sbi PORTC, RESET_PIN
33
34    ret
35
36
37  ; -------------------------------------------------------
38  ;               INTERRUPCIONES EXTERNAS
39  ; -------------------------------------------------------
40
41  config_int0:
42    ; Flanco negativo
43      ldi temp, (1 << ISC01) | (0 << ISC00)
44      sts EICRA, temp
45
46      ; Habilitar interrupci n
47    sbi EIMSK, INT0
48
49      ret
50
51
52  config_pci0:
53      ; Habilitar pin de ECHO
54    ldi temp, (1 << ULTRASOUND_ECHO)
55    sts PCMSK2, temp
56
57      ret
58
59
60  ; -------------------------------------------------------
61  ;                   COMUNICACI N SERIAL
62  ; -------------------------------------------------------
63
64  config_USART:
65      ; RX Complete Interrupt Enable, Enable TX & RX
66    ldi temp, (1 << RXCIE0) | (1 << RXEN0) | (1 << TXEN0)
67    sts UCSR0B, temp
68
69      ;               Asynchronous USART                   no-parity
                           8-bit data              1 stop bit
70    ldi temp, (0 << UMSEL01) | (0 << UMSEL00) | (0 << UPM01) | (0 <<
       UPM00) | (1 << UCSZ01) | (1 << UCSZ00) | (0 << USBS0)
71    sts UCSR0C, temp
```

```
72
73      ; Baud rate
74    ldi temp, LOW(UBRR0)
75    sts UBRR0L, temp
76    ldi temp, HIGH(UBRR0)
77    sts UBRR0H, temp
78
79    ret
80
81
82  ; -----------------------------------------------------
83  ;                         TIMERS
84  ; -----------------------------------------------------
85
86  config_timer0:
87    clr temp
88
89      ; Modo normal apagado
90    out TCCR0A, temp
91    out TCCR0B, temp
92
93    ; Interrupci n por overflow
94    ldi temp, (1 << TOIE0)
95    sts TIMSK0, temp
96
97    ret
98
99
100 config_timer1:
101   clr temp
102
103     ; Reiniciar cuenta del timer
104   sts TCNT1H, temp
105   sts TCNT1L, temp
106
107     ; Configurar la frecuencia del PWM
108   ldi temp, HIGH(TOP_PWM)
109   sts ICR1H, temp
110   ldi temp, LOW(TOP_PWM)
111   sts ICR1L, temp
112
113     ; Valores iniciales PWM
114     rcall actualizar_OCR1A
115     rcall actualizar_OCR1B
116
117     ; Set OC1A/OC1B on compare match when up-counting.
118     ; Clear OC1A/OC1B on compare match when down-counting.
119     ; Phase Correct PWM, top en ICR1
120   ldi temp, (1 << COM1A1) | (1 << COM1A0) | (1 << COM1B1) | (1 <<
      COM1B0) | (1 << WGM11) | (0 << WGM10)
121   sts TCCR1A, temp
122
123     ; Phase Correct PWM, top en ICR1, prescaler 1/8
124   ldi temp, (1 << WGM13) | (0 << WGM12) | (0 << CS12) | (1 << CS11)
      | (0 << CS10)
```

```
  sts TCCR1B, temp

  ret


config_timer2:
  clr temp

    ; Modo normal apagado
  sts TCCR2A, temp
  sts TCCR2B, temp

  ; Interrupci n por overflow
  ldi temp, (1 << TOIE2)
  sts TIMSK2, temp

  ret
```

```
; ---------------------------------
; MARK1_config.asm
;
; Created: 11/11/2023 11:49:48 AM
; Authors: FR & NN
; ---------------------------------


; -----------------------------------------------------
;                       PUERTOS
; -----------------------------------------------------

config_ports:
    ; Evitar pulso no deseado al comienzo del programa
  out PORTB, zero

  ; PORTB
  sbi DDRB,  SERVOA_PIN
    sbi DDRB,  SERVOB_PIN
    sbi DDRB,  ACTIVE_LED

  ; PORTD
  sbi DDRD,  LASER_PIN
    ; cbi DDRD,  INT0_PIN
    ; sbi PORTD, INT0_PIN
  cbi PORTD, ULTRASOUND_TRIG
  sbi DDRD,  ULTRASOUND_TRIG
  cbi DDRD,  ULTRASOUND_ECHO

  ; PORTC
  sbi DDRC,  RESET_PIN
  sbi PORTC, RESET_PIN

  ret


; -----------------------------------------------------
;               INTERRUPCIONES EXTERNAS
```

```
39 ; -----------------------------------------------------
40
41 config_int0:
42   ; Flanco negativo
43     ldi temp, (1 << ISC01) | (0 << ISC00)
44     sts EICRA, temp
45
46     ; Habilitar interrupci n
47   sbi EIMSK, INT0
48
49     ret
50
51
52 config_pci0:
53     ; Habilitar pin de ECHO
54   ldi temp, (1 << ULTRASOUND_ECHO)
55   sts PCMSK2, temp
56
57     ret
58
59
60 ; -----------------------------------------------------
61 ;                   COMUNICACI N SERIAL
62 ; -----------------------------------------------------
63
64 config_USART:
65     ; RX Complete Interrupt Enable, Enable TX & RX
66   ldi temp, (1 << RXCIE0) | (1 << RXEN0) | (1 << TXEN0)
67   sts UCSR0B, temp
68
69     ;               Asynchronous USART                    no-parity
                          8-bit data              1 stop bit
70   ldi temp, (0 << UMSEL01) | (0 << UMSEL00) | (0 << UPM01) | (0 <<
     UPM00) | (1 << UCSZ01) | (1 << UCSZ00) | (0 << USBS0)
71   sts UCSR0C, temp
72
73     ; Baud rate
74   ldi temp, LOW(UBRR0)
75   sts UBRR0L, temp
76   ldi temp, HIGH(UBRR0)
77   sts UBRR0H, temp
78
79   ret
80
81
82 ; -----------------------------------------------------
83 ;                          TIMERS
84 ; -----------------------------------------------------
85
86 config_timer0:
87   clr temp
88
89     ; Modo normal apagado
90   out TCCR0A, temp
91   out TCCR0B, temp
```

```
92
93    ; Interrupci n por overflow
94    ldi temp, (1 << TOIE0)
95    sts TIMSK0, temp
96
97    ret
98
99
100 config_timer1:
101   clr temp
102
103     ; Reiniciar cuenta del timer
104   sts TCNT1H, temp
105   sts TCNT1L, temp
106
107     ; Configurar la frecuencia del PWM
108   ldi temp, HIGH(TOP_PWM)
109   sts ICR1H, temp
110   ldi temp, LOW(TOP_PWM)
111   sts ICR1L, temp
112
113     ; Valores iniciales PWM
114     rcall actualizar_OCR1A
115     rcall actualizar_OCR1B
116
117     ; Set OC1A/OC1B on compare match when up-counting.
118     ; Clear OC1A/OC1B on compare match when down-counting.
119     ; Phase Correct PWM, top en ICR1
120   ldi temp, (1 << COM1A1) | (1 << COM1A0) | (1 << COM1B1) | (1 <<
      COM1B0) | (1 << WGM11) | (0 << WGM10)
121   sts TCCR1A, temp
122
123     ; Phase Correct PWM, top en ICR1, prescaler 1/8
124   ldi temp, (1 << WGM13) | (0 << WGM12) | (0 << CS12) | (1 << CS11)
      | (0 << CS10)
125   sts TCCR1B, temp
126
127   ret
128
129
130 config_timer2:
131   clr temp
132
133     ; Modo normal apagado
134   sts TCCR2A, temp
135   sts TCCR2B, temp
136
137   ; Interrupci n por overflow
138   ldi temp, (1 << TOIE2)
139   sts TIMSK2, temp
140
141   ret
```

```
1 ; ---------------------------------
2 ; MARK1_rutina.asm
3 ;
```

```
; Created: 11/11/2023 16:12:54
; Author: FR & NN
; ---------------------------------


; ------------------------------------------------------
;                        TIMER 0
; ------------------------------------------------------

start_timer0:
    ; Cuenta en 0
    clr temp
    out TCNT0, temp

    ; Prescaler 1024
    ldi temp, (1 << CS02) | (0 << CS01) | (1 << CS00)
    out TCCR0B, temp

    ret


stop_timer0:
    ; Apagado
    ldi temp, (0 << CS02) | (0 << CS01) | (0 << CS00)
    out TCCR0B, temp

    ret


; ------------------------------------------------------
;                        TIMER 2
; ------------------------------------------------------

start_timer2:
    ; Cuenta en 0
    clr temp
    sts TCNT2, temp

    ; Prescaler 8
    ldi temp, (0 << CS22) | (1 << CS21) | (0 << CS20)
  sts TCCR2B, temp
    ret


stop_timer2:
    ; Apagado
    ldi temp, (0 << CS22) | (0 << CS21) | (0 << CS20)
    sts TCCR2B, temp

    ret

; ------------------------------------------------------
;                        TRIGGER
; ------------------------------------------------------

```

```
send_trigger:
    ; No interrumpir para tener un pulso exacto siempre
    cli

  ; Limpiar flag de PCI2 y activar la interrupcion
    sbic PCIFR, PCIF2
  sbi PCIFR, PCIF2

  lds temp, PCICR
  ori temp, (1 << PCIE2)
  sts PCICR, temp

  sbi PORTD, ULTRASOUND_TRIG
  ldi temp, LOOPS_TRIGGER
  mov loop_index, temp
send_trigger_loop:
  dec loop_index
  brne send_trigger_loop
  cbi PORTD, ULTRASOUND_TRIG

    sei
  ret


; ------------------------------------------------------
;                COMUNICACI N SERIAL
; ------------------------------------------------------

; Se debe cargar previamente el registro data_type
send_data:
    cli

    ; Mandar primero el byte de tipo de dato
    mov temp_byte, data_type
    rcall send_byte

    ; Ver si hay que mandar bytes extra, seg n el tipo de dato

    cpi data_type, MEASUREMENT
    breq send_measurement

    cpi data_type, CURRENT_POSITION
    breq send_position

  cpi data_type, DEBUG
  breq send_debug

  cpi data_type, MEASURE_STATE
  breq send_state

  cpi data_type, INFO
  breq send_info

    ; Si se lleg  ac , no
    ; hace falta mandar bytes extra
```

```
114     rjmp send_data_end
115
116 send_measurement:
117   ; Convertir a cm primero.
118   rcall convertir_a_cm
119     ; Guardar en RAM el valor.
120     ; rcall convertir_lectura_ascii
121     ; Comentado porque es innecesario para nuestro proyecto.
122
123     ; Formato: STEPA, STEPB, LECTURAL, LECTURAH
124     ; (Little-endian)
125     mov temp_byte, stepa
126     rcall send_byte
127     mov temp_byte, stepb
128     rcall send_byte
129
130     mov temp_byte, lectural
131     rcall send_byte
132   mov temp_byte, lecturah
133     rcall send_byte
134
135     rjmp send_data_end
136
137 send_position:
138     ; Formato: STEPA, STEPB
139     mov temp_byte, stepa
140     rcall send_byte
141     mov temp_byte, stepb
142     rcall send_byte
143
144     rjmp send_data_end
145
146 send_debug:
147     ; Ac  poner el dato que sea necesario.
148   mov temp_byte, lecturah
149     rcall send_byte
150
151   rjmp send_data_end
152
153 send_state:
154   mov temp_byte, estado_medicion
155   rcall send_byte
156
157   rjmp send_data_end
158
159 send_info:
160   ; Leemos byte por byte en eeprom
161   ldy INFO_ADDR
162   ldi temp, MAX_STRING
163   mov loop_index, temp
164
165 send_info_loop:
166   rcall eeprom_read
167   rcall send_byte
168
```

```
169   cpi temp_byte, 0
170   breq send_info_loop_end
171
172   dec loop_index
173   breq send_info_loop_end
174
175   ld temp, y+ ; incrementar y
176   rjmp send_info_loop
177
178 send_info_loop_end:
179   rjmp send_data_end
180
181 send_data_end:
182     sei
183     ret
184
185
186 ; Se debe cargar previamente el registro temp_byte
187 send_byte:
188     lds temp, UCSR0A
189     sbrs temp, UDRE0
190     rjmp send_byte
191
192     sts UDR0, temp_byte
193     ret
194
195
196 ; -------------------------------------------------------
197 ;                 RUTINAS DE BYTES
198 ; -------------------------------------------------------
199
200 rutina_comando_byte_move_to:
201     ; Vemos a qu  corresponde este byte
202     mov temp, bytes_restantes
203
204     cpi temp, 2
205     breq comando_move_to_byte_stepa
206
207     cpi temp, 1
208     breq comando_move_to_byte_stepb
209
210     ; No deber amos llegar ac
211     rjmp rutina_comando_byte_move_to_end
212
213
214 comando_move_to_byte_stepa:
215     ; Todav a falta stepb...
216     mov stepa, byte_recibido
217     dec bytes_restantes
218     ldi estado_comando, WAIT_BYTE
219
220     rjmp rutina_comando_byte_move_to_end
221
222
223 comando_move_to_byte_stepb:
```

```
    ; Ya tenemos todos los datos! Podemos proceder
    mov stepb, byte_recibido

    ; Mover
    rcall actualizar_OCR1A
    rcall actualizar_OCR1B

    ; Notificar el cambio
    ldi data_type, CURRENT_POSITION
    rcall send_data

    ; Hacer un delay por overflows para
    ; dar tiempo al movimiento
    ldi left_ovfs, DELAY_MOVIMIENTO
    ldi estado_medicion, DELAY_MOVE_TO
    rcall start_timer0

    ; Ya podemos recibir nuevos comandos
    ldi estado_comando, WAIT_COMMAND

rutina_comando_byte_move_to_end:
    ret









rutina_comando_byte_scan_region:
    mov temp, bytes_restantes

    cpi temp, 4
    breq comando_byte_first_stepa

    cpi temp, 3
    breq comando_byte_first_stepb

  cpi temp, 2
  breq comando_byte_last_stepa

  cpi temp, 1
  breq comando_byte_last_stepb

comando_byte_first_stepa:
  mov first_stepa, byte_recibido
    dec bytes_restantes
    ldi estado_comando, WAIT_BYTE

  rjmp rutina_comando_byte_scan_region_end

comando_byte_first_stepb:
  mov first_stepb, byte_recibido
    dec bytes_restantes
```

```
279     ldi estado_comando, WAIT_BYTE
280
281   rjmp rutina_comando_byte_scan_region_end
282
283 comando_byte_last_stepa:
284   mov last_stepa, byte_recibido
285     dec bytes_restantes
286     ldi estado_comando, WAIT_BYTE
287
288   rjmp rutina_comando_byte_scan_region_end
289
290 comando_byte_last_stepb:
291     ; Ya tenemos todos los datos
292   mov last_stepb, byte_recibido
293   rcall start_scan
294
295     ; Podemos recibir nuevos comandos
296     ldi estado_comando, WAIT_COMMAND
297
298   rjmp rutina_comando_byte_scan_region_end
299
300 rutina_comando_byte_scan_region_end:
301     ret
302
303
304
305
306
307
308
309
310 rutina_comando_byte_write_info:
311     ; Guardar al b ffer
312     st x+, byte_recibido
313     ldi estado_comando, WAIT_BYTE
314
315     ; Fijarse si es el null para terminar
316     cpi byte_recibido, 0
317     breq rutina_comando_byte_write_info_eeprom
318
319     ; Fijarse si llegamos al l mite de
320     ; longitud del string
321     dec bytes_restantes
322     breq rutina_comando_byte_write_info_eeprom
323
324     rjmp rutina_comando_byte_write_info_end
325
326 rutina_comando_byte_write_info_eeprom:
327     ; Iniciar escritura de RAM a EEPROM
328     ; Bloquea el programa
329     rcall copiar_buffer_a_eeprom
330     ldi estado_comando, WAIT_COMMAND
331
332     ; Notificar escritura completa
333     ldi data_type, WRITE_INFO_DONE
```

```
334     rcall send_data
335
336 rutina_comando_byte_write_info_end:
337     ret
338
339
340 ; ------------------------------------------------------
341 ;                 RUTINAS DE COMANDOS
342 ; ------------------------------------------------------
343
344 rutina_comando_abort:
345     ; Nos quedamos donde estamos
346     ldi estado_medicion, WAIT_MEDIR
347     ldi estado_comando, WAIT_COMMAND
348
349     ret
350
351
352 rutina_comando_ping:
353     ; Ping - pong
354     ldi data_type, PONG
355     rcall send_data
356   ldi estado_comando, WAIT_COMMAND
357
358     ret
359
360
361 rutina_comando_ask_position:
362     ; Devolvemos la posici n
363     ldi data_type, CURRENT_POSITION
364     rcall send_data
365   ldi estado_comando, WAIT_COMMAND
366
367     ret
368
369
370 rutina_comando_ask_laser:
371     ; Devolvemos el estado actual del l ser
372     sbis PORTD, LASER_PIN
373     ldi data_type, LASER_OFF
374     sbic PORTD, LASER_PIN
375     ldi data_type, LASER_ON
376     rcall send_data
377   ldi estado_comando, WAIT_COMMAND
378
379     ret
380
381
382 rutina_comando_ask_state:
383   ldi data_type, MEASURE_STATE
384   rcall send_data
385   ldi estado_comando, WAIT_COMMAND
386
387   ret
388
```

```
389
390 rutina_comando_ask_info:
391   ldi data_type, INFO
392   rcall send_data
393   ldi estado_comando, WAIT_COMMAND
394
395   ret
396
397
398 rutina_comando_scan_row:
399   mov first_stepa, zero
400   ldi temp, MAX_STEPA
401   mov last_stepa, temp
402
403   mov first_stepb, stepb
404   mov last_stepb, stepb
405
406   rcall start_scan
407     ldi estado_comando, WAIT_COMMAND
408
409   ret
410
411
412 rutina_comando_scan_col:
413   mov first_stepa, stepa
414   mov last_stepa, stepa
415
416   mov first_stepb, zero
417   ldi temp, MAX_STEPB
418   mov last_stepb, temp
419
420   rcall start_scan
421     ldi estado_comando, WAIT_COMMAND
422
423   ret
424
425
426 rutina_comando_scan_all:
427   mov first_stepa, zero
428   ldi temp, MAX_STEPA
429   mov last_stepa, temp
430
431   mov first_stepb, zero
432   ldi temp, MAX_STEPB
433   mov last_stepb, temp
434
435   rcall start_scan
436     ldi estado_comando, WAIT_COMMAND
437
438   ret
439
440
441 rutina_comando_scan_region:
442     ; Necesitamos 4 bytes mas (first_stepa, first_stepb, last_stepa
    , last_stepb)
```

```
443   ldi temp, 4
444   mov bytes_restantes, temp
445   ldi estado_comando, WAIT_BYTE
446 
447   ret
448 
449 
450 rutina_comando_move_to:
451     ; Necesitamos 2 bytes mas (stepa, stepb)
452   ldi temp, 2
453   mov bytes_restantes, temp
454     ldi estado_comando, WAIT_BYTE
455 
456     ret
457 
458 
459 rutina_comando_medir_dist:
460     mov first_stepa, stepa
461   mov last_stepa, stepa
462     mov first_stepb, stepb
463   mov last_stepb, stepb
464 
465     ; Apagar el laser
466     cbi PORTD, LASER_PIN
467     ldi data_type, LASER_OFF
468     rcall send_data
469 
470   ; Setear la distancia m nima en 0xFFFF
471   clr min_disth
472   dec min_disth
473 
474   clr min_distl
475   dec min_distl
476 
477     ldi estado_medicion, MEDIR
478 
479     ldi estado_comando, WAIT_COMMAND
480 
481     ret
482 
483 
484 rutina_comando_turn_on_laser:
485   ; Encender l ser y notificar cambio de estado
486   sbi PORTD, LASER_PIN
487     ldi data_type, LASER_ON
488     rcall send_data
489   ldi estado_comando, WAIT_COMMAND
490 
491     ret
492 
493 
494 rutina_comando_turn_off_laser:
495   ; Apagar l ser y notificar cambio de estado
496   cbi PORTD, LASER_PIN
497     ldi data_type, LASER_OFF
```

```
498     rcall send_data
499   ldi estado_comando, WAIT_COMMAND
500 
501     ret
502 
503 
504 rutina_comando_write_info:
505     ldi temp, MAX_STRING
506   mov bytes_restantes, temp
507     ldx buffer
508     ldi estado_comando, WAIT_BYTE
509 
510     ret
511 
512 ; ------------------------------------------------------
513 ;                       START SCAN
514 ; ------------------------------------------------------
515 
516 start_scan:
517   ; Deben estar cargados los valores
518     ; l mite de stepa y stepb seg n sea el caso
519 
520   ; Mover el servo A y B a la primer posici n
521   mov stepa, first_stepa
522   mov stepb, first_stepb
523 
524   rcall actualizar_OCR1A
525   rcall actualizar_OCR1B
526 
527     ; Apagar el laser
528     cbi PORTD, LASER_PIN
529     ldi data_type, LASER_OFF
530     rcall send_data
531 
532     ; Notificar cambio de posici n
533   ldi data_type, CURRENT_POSITION
534     rcall send_data
535 
536   ; Hacer un delay por overflows para
537     ; dar tiempo al movimiento
538     ldi left_ovfs, DELAY_MOVIMIENTO
539     ldi estado_medicion, DELAY_SCAN
540     rcall start_timer0
541 
542   ; Setear la distancia m nima en 0xFFFF
543   clr min_disth
544   dec min_disth
545 
546   clr min_distl
547   dec min_distl
548 
549   ret
550 
551 
552 ; ------------------------------------------------------
```

```
553 ;                           EEPROM
554 ; -----------------------------------------------------
555
556 copiar_buffer_a_eeprom:
557     ldx buffer
558     ldy INFO_ADDR
559     ldi temp, MAX_STRING
560     mov loop_index, temp
561
562 copiar_buffer_a_eeprom_loop:
563     ld temp_byte, x+
564     rcall eeprom_write
565     ld temp, y+  ; Incrementar Y
566
567     cpi temp_byte, 0
568     breq copiar_buffer_a_eeprom_end
569
570     dec loop_index
571     breq copiar_buffer_a_eeprom_end
572
573     rjmp copiar_buffer_a_eeprom_loop
574
575 copiar_buffer_a_eeprom_end:
576     ; Por las dudas, finalizar con un NULL
577     ; en la  ltima  posici n escrita
578     ld temp, -y
579     clr temp_byte
580     rcall eeprom_write
581
582     ret
583
584
585 ; Usa temp_byte y el puntero Y
586 ; para la direcci n
587 eeprom_write:
588     ; Verificar si podemos escribir
589     sbic EECR, EEPE
590     rjmp eeprom_write
591
592     ; Address
593     out EEARH, yh
594     out EEARL, yl
595
596     ; Data
597     out EEDR, temp_byte
598
599     ; Enable
600     sbi EECR, EEMPE
601     sbi EECR, EEPE
602
603     ret
604
605
606 ; Usa temp_byte y el puntero Y
607 ; para la direcci n
```

```
608 eeprom_read:
609   ; Esperar a una posible escritura en curso
610   sbic EECR, EEPE
611   rjmp eeprom_read
612 
613   ; Address de lectura
614   out EEARH, yh
615   out EEARL, yl
616 
617   ; Iniciar lectura
618   sbi EECR, EERE
619 
620   ; Byte leido
621   in temp_byte, EEDR
622 
623   ret
624 
625 
626 ; ---------------------------------------------------------
627 ;                   STEP UPS Y DOWNS
628 ; ---------------------------------------------------------
629 
630 ; Incrementa STEPA si es posible
631 stepa_up:
632     cpi stepa, MAX_STEPA
633     breq stepa_up_end
634 
635     inc stepa
636     rcall actualizar_OCR1A
637 
638 stepa_up_end:
639     ret
640 
641 
642 ; Decrementa STEPA si es posible
643 stepa_down:
644     cpi stepa, 0
645     breq stepa_down_end
646 
647     dec stepa
648     rcall actualizar_OCR1A
649 
650 stepa_down_end:
651     ret
652 
653 
654 ; Incrementa STEPB si es posible
655 stepb_up:
656     cpi stepb, MAX_STEPB
657     breq stepb_up_end
658 
659     inc stepb
660     rcall actualizar_OCR1B
661 
662 stepb_up_end:
```

```
663     ret
664
665
666 ; Decrementa STEPB si es posible
667 stepb_down:
668     cpi stepb, 0
669     breq stepb_down_end
670
671     dec stepb
672     rcall actualizar_OCR1B
673
674 stepb_down_end:
675     ret
676
677
678 ; -----------------------------------------------------
679 ;                 MODIFICACI N DE OCR1X
680 ; -----------------------------------------------------
681
682 ; Las funciones step up, down lo hacen autom ticamente
683 ; Escribe en OCR1A = STEPA * STEP_OCR1A + MIN_OCR1A
684 actualizar_OCR1A:
685   push r0
686   push r1
687
688     in temp, sreg
689     push temp
690     cli
691
692     ; Multiplicaci n
693     ldi temp, STEP_OCR1A
694     mul stepa, temp
695
696     ; Suma
697     ldi xl, LOW(MIN_OCR1A)
698     ldi xh, HIGH(MIN_OCR1A)
699     add xl, r0
700     adc xh, r1
701
702     ; Guardado
703     sts OCR1AH, xh
704     sts OCR1AL, xl
705
706     pop temp
707     out sreg, temp
708
709   pop r1
710   pop r0
711
712     ret
713
714
715 ; Las funciones step up, down lo hacen autom ticamente
716 ; Escribe en OCR1B = STEPB * STEP_OCR1B + MIN_OCR1B
717 actualizar_OCR1B:
```

```
718   push r0
719   push r1
720
721     in temp, sreg
722     push temp
723     cli
724
725     ldi temp, STEP_OCR1B
726     mul stepb, temp
727
728     ldi xl, LOW(MIN_OCR1B)
729     ldi xh, HIGH(MIN_OCR1B)
730
731     add xl, r0
732     adc xh, r1
733
734     sts OCR1BH, xh
735     sts OCR1BL, xl
736
737     pop temp
738     out sreg, temp
739
740   pop r1
741   pop r0
742     ret
743
744 ; ----------------------------------------------------
745 ;                       ARITM TICA
746 ; ----------------------------------------------------
747
748
749 ; Convierte los bytes lecturah:lectural a centimetros
750 ; mediante una division
751 convertir_a_cm:
752
753   ; Divisor
754   push r18
755   push r19
756   ldi r18, low(DIVISOR_CONVERSION)
757   ldi r19, high(DIVISOR_CONVERSION)
758
759
760   ; Dividendo
761   push r16
762   push r17
763     mov r16, lectural
764   mov r17, lecturah
765
766   rcall division_16bits
767
768   ; Cociente
769   mov lectural, r16
770   mov lecturah, r17
771
772   pop r17
```

```
  pop r16
  pop r19
  pop r18

  ret


; Fijarse los registros que utiliza
division_16bits:
    ; Algoritmo basado en:
    ; https://www.microchip.com/en-us/application-notes?rv=1234aaef

    ; Resto
    push r14 ; Low
    push r15 ; High

    ; Cociente y Dividendo: r16 y r17

    ; Divisor: r18 y r19 (low, high)

    ; Contador auxiliar
    push r20

    ; Reiniciar resto y bit de carry
    clr r14
  sub r15, r15 ; Esto borra el carry
  ldi r20, 17 ; El loop itera una vez por bit

division_16bits_1:
    ; Tomar el bit m s significativo del dividendo
    ; y guardarlo en el carry
    rol r16
  rol r17

    ; Si ya recorrimos todos los bits terminamos
    dec r20
  brne division_16bits_2
    rjmp division_16bits_end

division_16bits_2:
    ; Mover el bit guardado en el carry a la posici n
    ; menos significativa del resto
    rol r14
  rol r15

    ; Vemos si podemos restar un divisor al resto
    ; actual
  sub r14, r18
  sbc r15, r19

    ; Si no se puede, volvemos a dejar el resto como estaba
    ; Y ponemos un 0 en el resultado final (va al carry
    ; y luego al registro r16)
  brcc division_16bits_3
  add r14, r18
```

```
828   adc r15, r19
829   clc
830   rjmp division_16bits_1
831
832     ; Si se pudo restar, en el resultado guardarmos un 1.
833 division_16bits_3:
834     sec
835   rjmp division_16bits_1
836
837 division_16bits_end:
838     pop r20
839     pop r15
840     pop r14
841     ret
842
843 ; Convierte la mindisth:mindistl (16 bits)
844 ; a un string ASCII en RAM
845 convertir_lectura_ascii:
846     ldx lectura_ascii
847
848     ; Nibble m s significativo 0x?...
849     mov temp_byte, lecturah
850     andi temp_byte, 0xF0
851     swap temp_byte
852     rcall convertir_byte_ascii
853     st x+, temp_byte
854
855     ; Nibble 0x.?..
856     mov temp_byte, lecturah
857     andi temp_byte, 0x0F
858     rcall convertir_byte_ascii
859     st x+, temp_byte
860
861     ; Nibble 0x..?.
862     mov temp_byte, lectural
863     andi temp_byte, 0xF0
864     swap temp_byte
865     rcall convertir_byte_ascii
866     st x+, temp_byte
867
868     ; Nibble menos significativo 0x...?
869     mov temp_byte, lectural
870     andi temp_byte, 0x0F
871     rcall convertir_byte_ascii
872     st x+, temp_byte
873
874     ret
875
876 ; Convierte el byte en temp_byte a su d gito ASCII
877 convertir_byte_ascii:
878     ; Para valores 0-9, sumar 0x30
879     cpi temp_byte, 9
880     brlo convertir_byte_sumar_30
881
882     ; Sino, sumar 0x41 - 0x0A
```

```
883     ldi temp, 0x41 - 0x0A
884     add temp_byte, temp
885 
886     rjmp convertir_byte_ascii_end
887 
888 convertir_byte_sumar_30:
889     ldi temp, 0x30
890     add temp_byte, temp
891 
892 convertir_byte_ascii_end:
893     ret
```