

Proyecto:

iRoboPong5000

Profesor:	Ing. Guillermo Campiglio
Cuatrimestre / Año:	2do Cuatrimestre de 2018
Turno de clases prácticas:	Miércoles
Jefe de Trabajos Prácticos:	Ricardo Arias
Docente guía:	Fabricio Baglivo

Autores			Seguimiento del proyecto								
Nombre	Apellido	Padrón									
Axel Brian	Erlich	94847									

Observaciones:

Fecha de aprobación			Firma J.T.P.

COLOQUIO	
Nota final	
Firma Profesor	

Contenido

Contenido	2
Introducción	3
Descripción General	4
Estructura general de diseño	6
Modo Automático	7
Modo Manual	8
Control de motores	9
Control de PWM para motores DC	9
Control de PWM para Servos	11
Comunicación Serial	13
Comunicación Serial: Proveedor	13
Descripción de Hardware	14
Diagrama Esquemático	15
Conclusiones	16
Anexo 1: Comandos	17
Anexo 2: Ejercicios Automáticos	19
Modo 1	19
Modo 2	20
Modo 3	20
Anexo 3: Componentes y costos	21
Componentes electrónicos	21
Componentes mecánicos	22
Anexo 4: Código	22

Introducción

En este documento se describe el proceso de desarrollo, armado y funcionamiento del iRoboPong5000, para la materia Laboratorio de microcontroladores (6609) de la facultad de Ingeniería de la Universidad de Buenos Aires.

El proyecto se desarrolló utilizando un Arduino Mega, programando el microcontrolador AtMega2560 en código avr assembly.

La principal función del iRoboPong5000 es la de permitir a un jugador de tenis de mesa, entrenar sin un compañero, el robot sirve como lanzador de pelotitas con distintas configuraciones que se explicarán a lo largo de este documento.

Con este robot, se logra una simulación muy cercana a las pelotitas que se recibirán en el juego con un compañero.

Además permite una ejercitación constante, ya que posee un almacenamiento de pelotitas de tamaño considerable lo cual permite ejercitar por varios minutos sin necesidad de parar.

Descripción General

iRoboPong5000 es un robot lanzador de pelotitas de ping-pong similar a los que se pueden encontrar en el mercado, pero con algunos aspectos extras que permitirán comportamientos personalizables según los requerimientos del usuario.



Imagen 1: Robot comercial HUI PANG HP-07

En la imagen 1 se puede ver un robot comercializado por la empresa HUI PANG y en la imagen 2, el robot desarrollado sobre el que se basa este documento.

El robot tiene dos modos de funcionamiento:

1. Automático
2. Manual

Se describe cada modo de funcionamiento con más detalle en las secciones correspondientes.

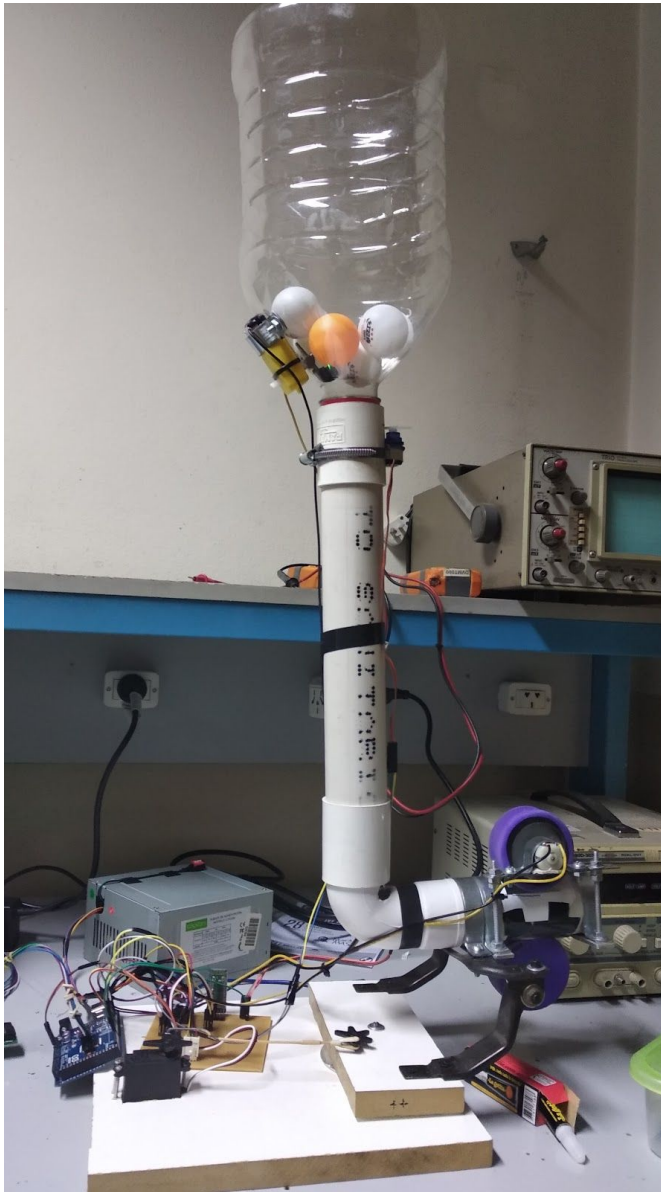


Imagen 2: Robot desarrollado

El control del robot es completamente inalámbrico a través de un Smartphone con conexión bluetooth.

Estructura general de diseño

El robot se controla en función de una comunicación serial a través de un módulo bluetooth en donde el mismo recibe comandos idealmente desde un teléfono celular. En caso de necesitar configurar un dispositivo para la comunicación, la lista de comando a la cual responde se puede ver en el anexo 1.

En la imagen 3 se muestra un diagrama de estados de funcionamiento del robot, las transiciones entre los estados se darán al recibir un comando válido.

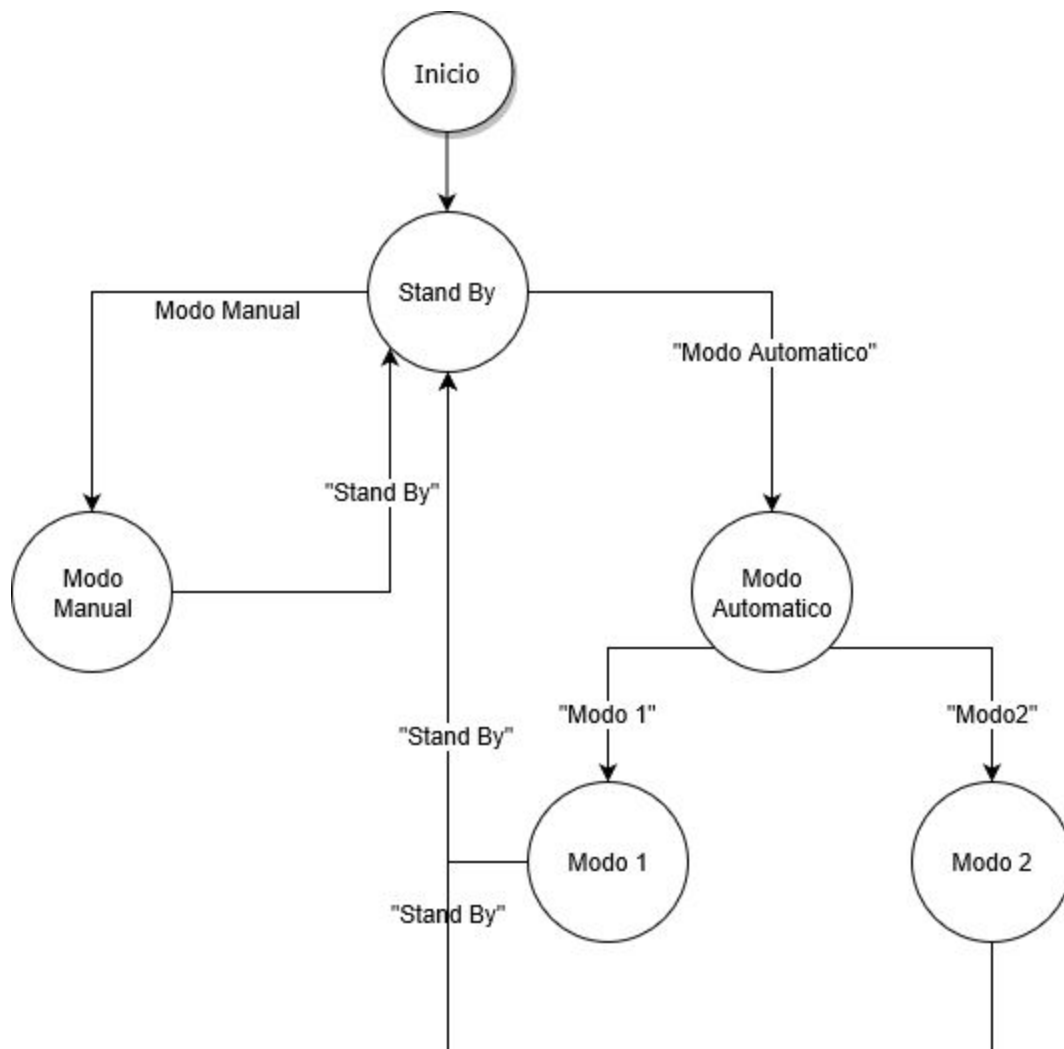


Imagen 3: Diagrama de estados de alto nivel del programa

Modo Automático

El modo más simple de uso es el modo automatico, lo unico se le debe indicar al robot es el ejercicio que debe ejecutar, los ejercicios vienen precargados en la programación.

Los ejercicios automáticos que vienen precargados son las rutinas más cotidianas que se utiliza en el deporte para ejercitar distintos aspectos del juego.

Ver los anexos 1 y 2 para más detalles sobre los comandos para ejecutar el modo automatico y cuales son las rutinas que se ejecutan.

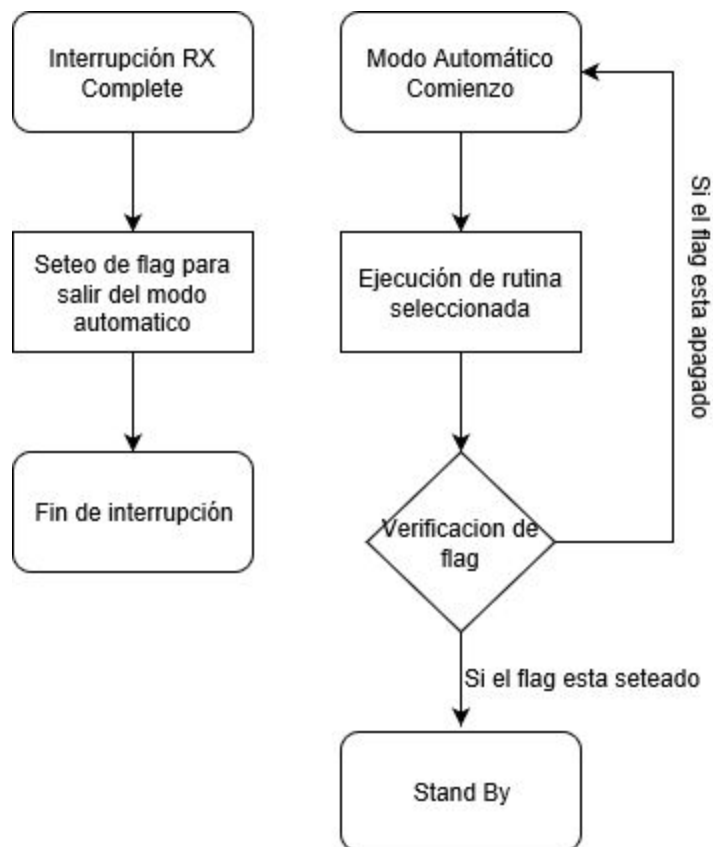


Imagen 4: Flujo de los modos automáticos

Modo Manual

En el modo manual, se pueden configurar distintos parámetros del robot por separado para lograr una variedad de tiros mucho mayor a los del modo automático, específicamente se pueden setear la velocidad de disparo, el ángulo de disparo y el momento en el cual disparar, cabe aclarar que este modo es para usar entre dos personas, mientras una entrena, la otra controla el robot.

Ver el anexo 1 para más detalles sobre los comandos para configurar cada parámetro.

En el modo manual, el microcontrolador, ingresa en el modo sleep hasta recibir un comando, esto hace que se active la interrupción de recepción completa, luego se lee el dato recibido y se ejecuta el código correspondiente en función del mismo.

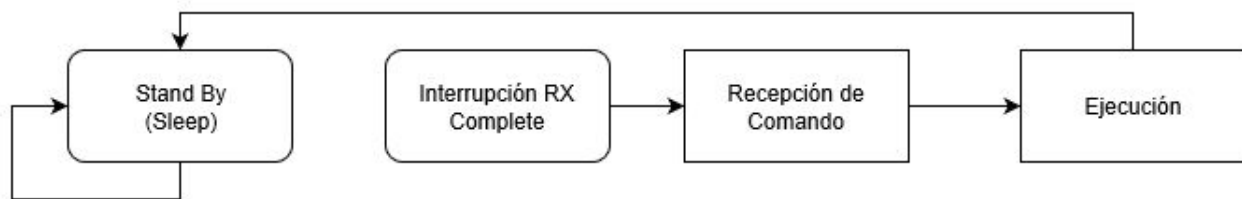


Imagen 5: Flujo en el modo manual

Control de motores

En las secciones siguientes se verá la parte técnica del desarrollo.

Control de PWM para motores DC

Los motores DC son controlados poniendo en uno de sus bornes una señal PWM, esta señal se logra aprovechando el modo de funcionamiento 'Fast PWM' de los distintos timers que posee el microcontrolador.

En este caso, se utiliza el oscilador de 16MHz del ATmega2560 para alimentar un timer del mismo a una frecuencia de aproximadamente 62,5 KHz y para lograr distintas velocidades, se va cambiando el registro comparador para así construir distintos duty cycles.

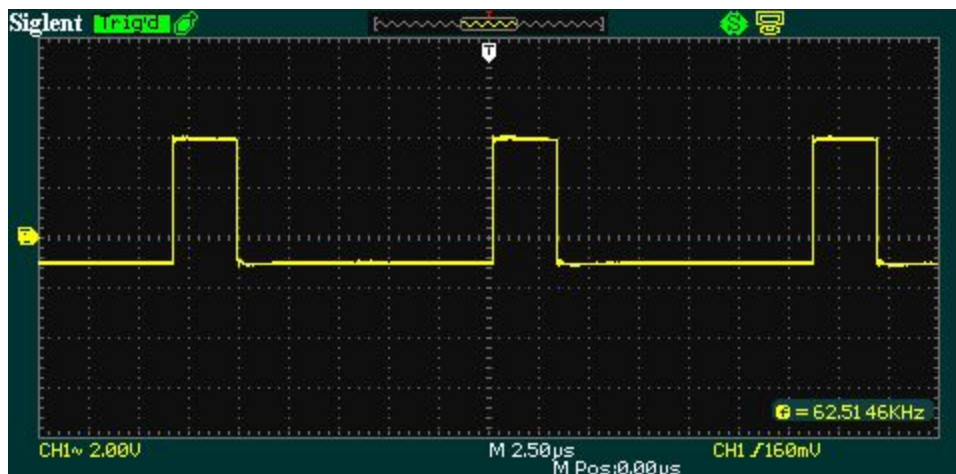


Imagen 6: PWM para control de motor DC

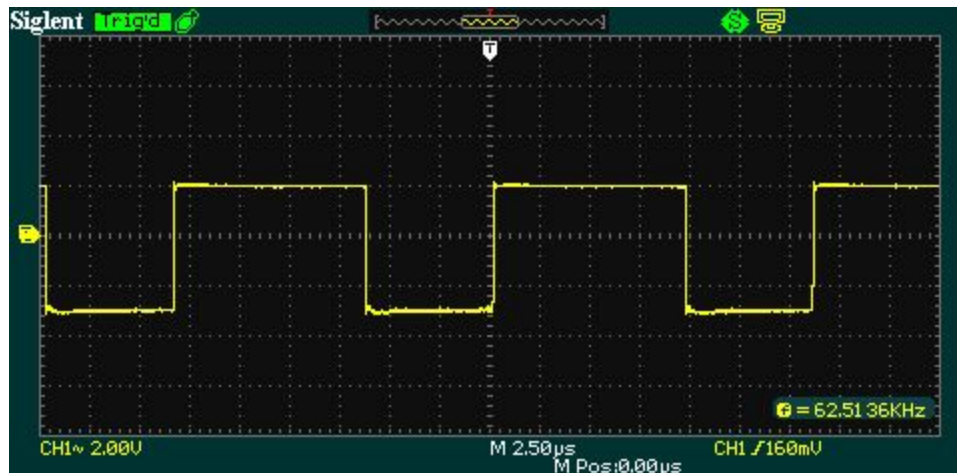


Imagen 7: PWM para control de motor DC

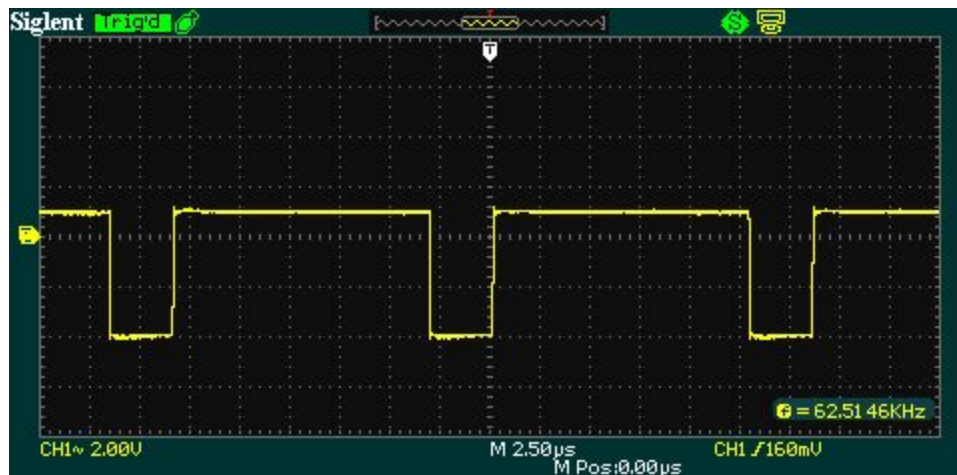


Imagen 8: PWM para control de motor DC

Cuando hay un match entre la cuenta del timer y el registro comparador, la señal se apaga. La trampa está en los motores disparadores, en cuyo caso el motor debe poder configurarse para que funcione en ambos sentidos de giro, para lograr esto lo que se hace es invertir el otro borne del motor (si estaba en 0v se manda a Vin, si estaba en Vin se manda a 0v) y a la vez complementar el PWM para mantener la velocidad que se tenía antes de invertir el sentido de giro.

Control de PWM para Servos

El caso de los Servos es similar a los motores DC pero con una vuelta de tuerca extra, los servos reciben una señal de control por PWM pero esta señal debe estar en un rango de frecuencia particular, además para moverse no utilizan todo el rango disponible de duty cycle (0-100%) sino que utilizan un rango más pequeño que define el ángulo de giro (5-10%) de 180 grados.

Para controlarlos entonces se utilizó un prescaler sobre la señal del timer para lograr una frecuencia aproximada de 60 Hz, además se fueron probando distintos valores para el registro comparador para poder lograr la máxima amplitud de movimiento del servo.

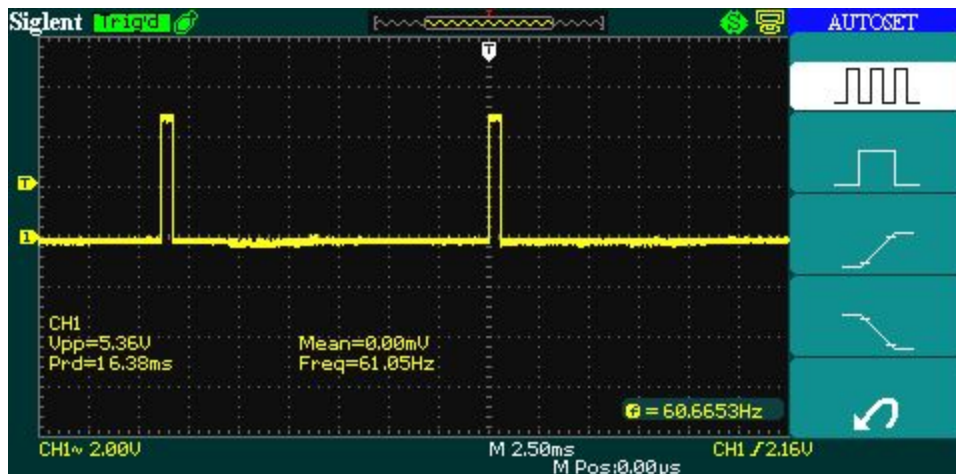


Imagen 9: PWM para control de servomotor

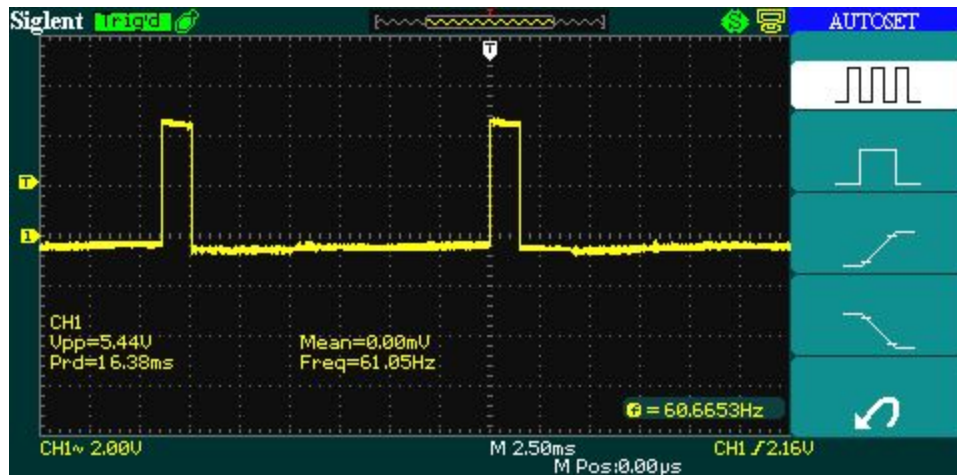


Imagen 10: PWM para control de servomotor

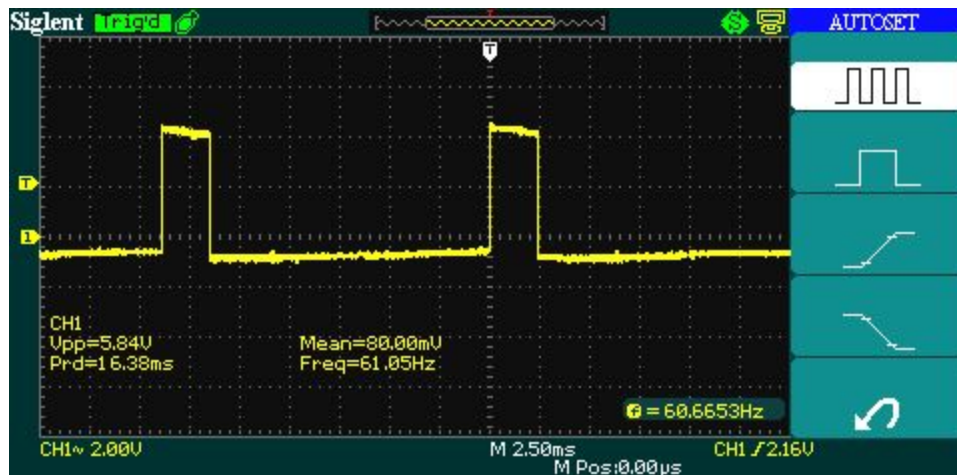


Imagen 11: PWM para control de servomotor

Comunicación Serial

Para la comunicación se utilizó el módulo USART del microcontrolador para establecer una conexión con un dispositivo móvil.

La comunicación se configuró en modo asincrónico con un baud rate de 9600.

Comunicación Serial: Proveedor

Por parte del dispositivo móvil, se establece la comunicación mediante bluetooth con el modulo HC-05 y para poder enviar los comandos se utiliza cualquier aplicación genérica de bluetooth disponibles en Play Store, se aclara que se probó con un dispositivo Android pero los principios deberían ser los mismo para un dispositivo iOS.

Ver el Anexo 1 para poder configurar la aplicación con los comandos que espera recibir el robot.

Descripción de Hardware

A continuación un diagrama en bloques del hardware utilizado:

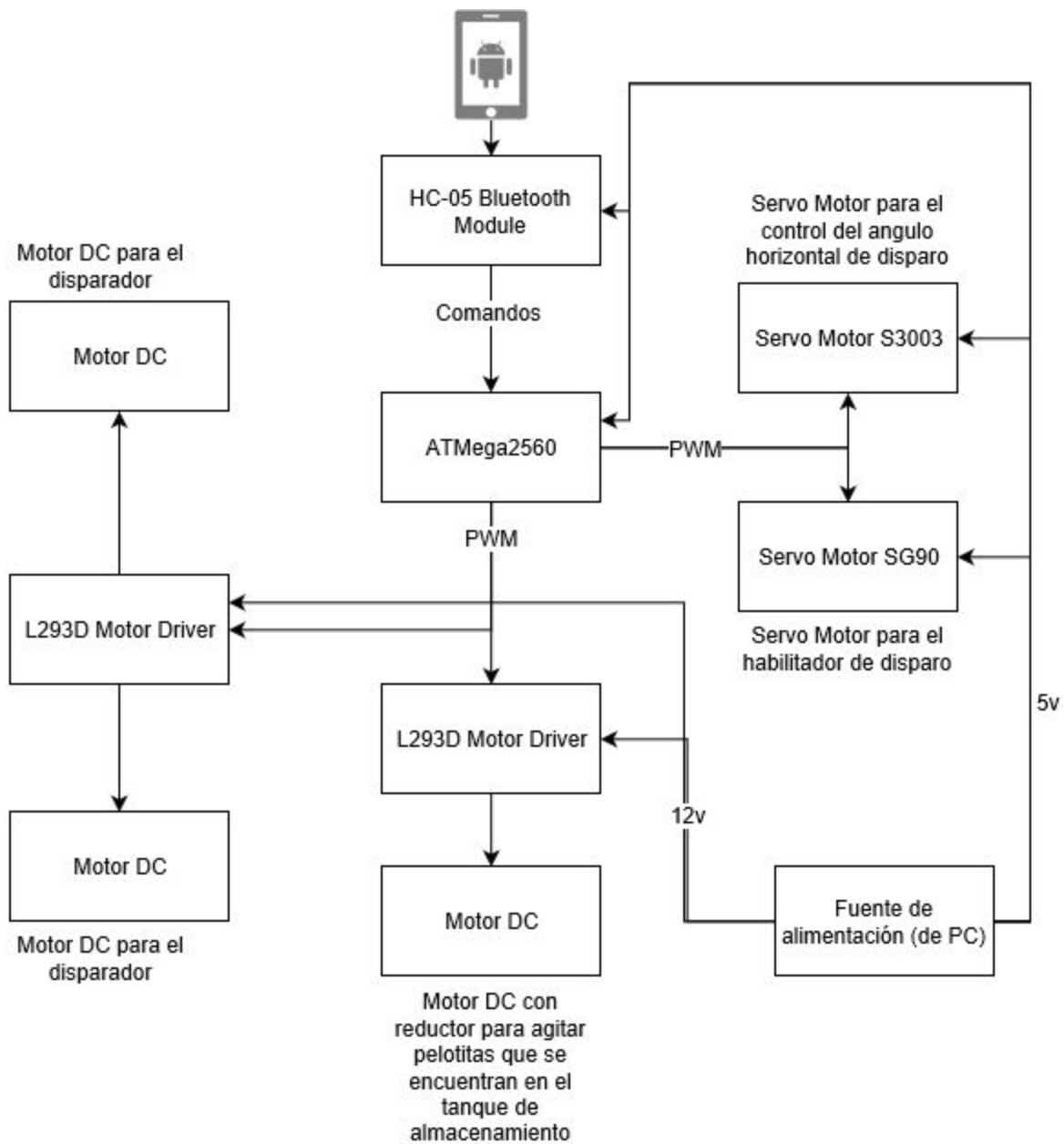


Imagen 12: diagrama general de bloques

Como se puede observar, se utilizan dos motores DC para disparar además de uno para agitar las pelotitas, también se utilizan dos servomotores, uno para el movimiento horizontal y otro para la habilitación de disparo.

Diagrama Esquemático

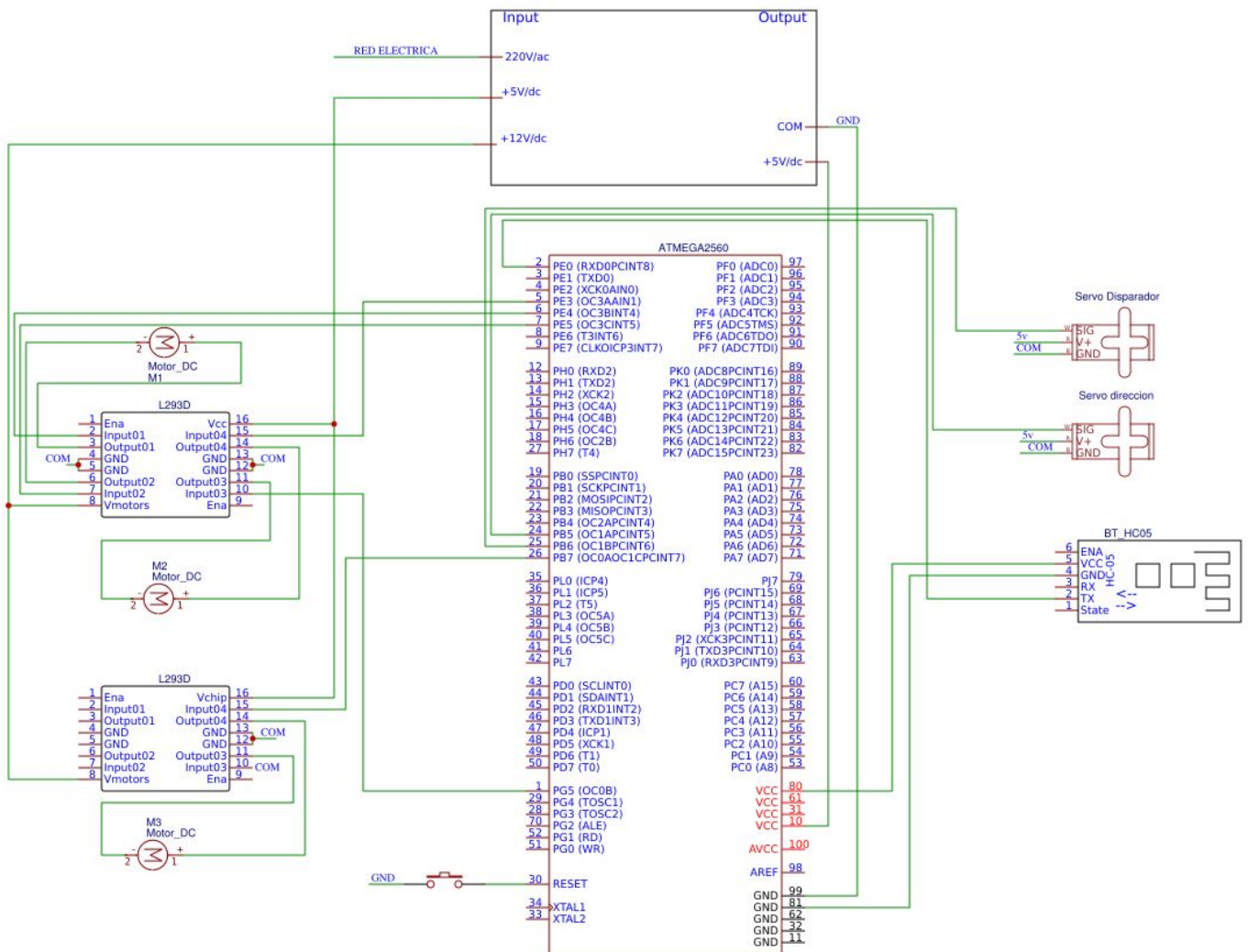


Imagen 13: Diagrama esquemático del proyecto

Conclusiones

En el desarrollo del proyecto, se tuvo que utilizar distintos módulos del microcontrolador ATmega2560, los módulos aplicados fueron diferentes timers y comunicación serial USART, además, por primera vez se tuvo la experiencia de programar un microcontrolador a bajo nivel con fines prácticos.

A nivel mecánico, se intentó obtener un resultado lo más realista posible dentro de un presupuesto reducido para invertir en hardware, la idea fue obtener un modelo “low cost” de las costosas versiones comerciales disponibles en el mercado.

La principal ventaja del proyecto es la posibilidad de poder programar rutinas de entrenamiento completamente personalizables, lo cual no es posible en los modelos comerciales, que cuales vienen con rutinas precargadas y no se pueden modificar. Las distintas rutinas de entrenamiento se encuentran programadas de forma estática, un punto a mejorar sería darle al usuario final la posibilidad de configurarlas a través de una interfaz de usuario.

Otro punto a mejorar es la falta de versatilidad en los motores que controlan el disparo, ya que son motores que generan muchas revoluciones pero poco torque por lo que la variación de velocidades no es precisa ni rápida.

El resultado es un dispositivo que cumple con los objetivos planteados, con margen para futuras mejoras, pero que podría ser muy útil en el uso para entrenamientos diarios.

Anexo 1: Comandos

En este anexo se describen los comandos que dirigen el funcionamiento del robot

Comando	Comentario	Acción
s		Ir al estado 'stand by'
a + q	q entero entre 1 y 3	Ir al modo automático 'q'
b		Configurar el disparo con backspin
t		Configurar el disparo con topspin
v + q	q entero entre 0 y 5	Configurar la velocidad de disparo
n + q	q entero entre 1 y 5	Configurar el ángulo de disparo
d		Disparar
m		Arrancar el mezclador de pelotitas

Tabla 1: Lista de comandos

Con estos comandos se podrá controlar el robot, ya sea enviando los comandos a través de una terminal al puerto serie del microcontrolador, o configurando una aplicación propia para que envíe los comandos correspondientes al presionar botones.

Se encontro una aplicacion en la play store ideal para manejar el robot, la cual permite configurar distintos controles para que mandes los comandos correspondientes para controlar el robot. El nombre de la aplicación es "Bluetooth Electronics". En la imagen 13 se puede observar una captura de pantalla con los controles.

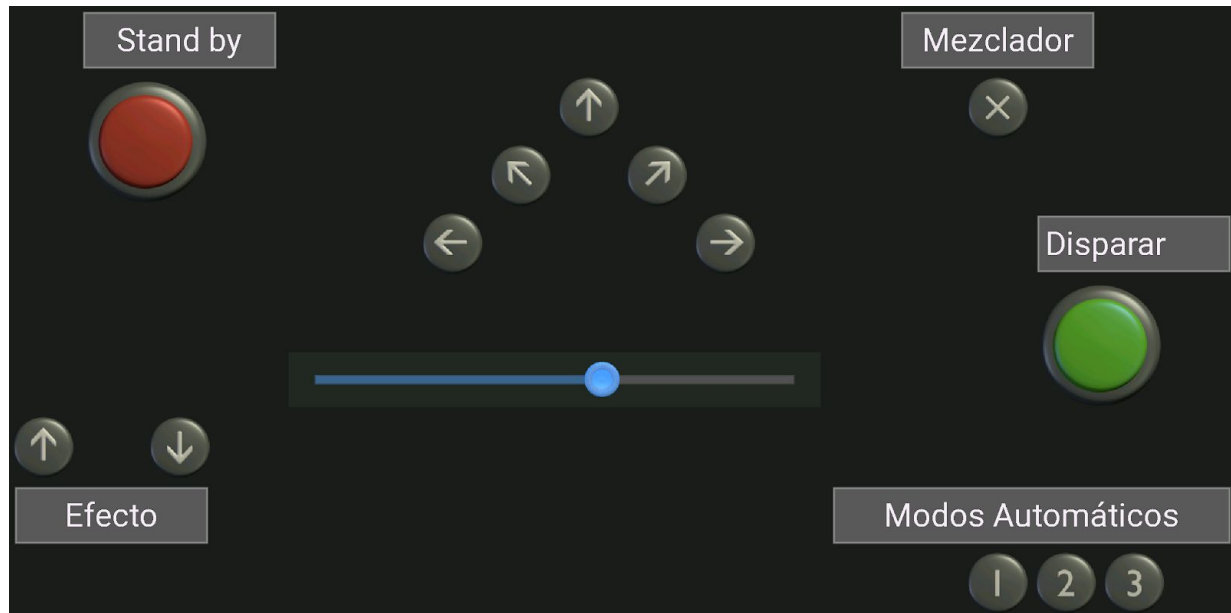


Imagen 14: Pantalla de la aplicación para controlar el robot.

Anexo 2: Ejercicios Automáticos

El robot posee tres modos automáticos preprogramados, a continuación una breve descripción de las acciones que realiza en cada modo:

Modo 1

Este modo se ejecuta con la máxima velocidad de disparo, se ejecutan los siguientes pasos en un loop infinito o hasta cambiar de modo. Es el ejercicio más clásico para trabajar la movilidad de piernas en el deporte.

Acciones en orden de ejecución
Disparo a la derecha
Delay 3/4 de segundo
Disparo al centro
Delay 3/4 de segundo
Disparo a la derecha
Delay 3/4 de segundo
Disparo a la izquierda

Tabla 2: Modo automático 1

Modo 2

Este ejercicio sirve para ejercitar golpes de revés, corte, drive y contratop.

Acciones en orden de ejecución
Disparo corto a la izquierda
Delay 1 segundo
Disparo medio a la izquierda
Delay 1 segundo
Disparo largo a la izquierda

Tabla 3: Modo automático 2

Modo 3

En este modo se ejercitan los distintos golpes de derecha, corte, drive, y contratop.

Acciones en orden de ejecución
Disparo corto a la derecha
Delay 1 segundo
Disparo medio a la derecha
Delay 1 segundo
Disparo largo a la derecha

Tabla 4: Modo automático 3

Anexo 3: Componentes y costos

En este anexo se enumeran los componentes del proyecto y su costo aproximado:

Componentes electrónicos

	Costo	Cantidad	Total
L293D Motor Driver	\$150.00	2	\$300.00
Arduino Mega	\$600.00	1	\$600.00
Programador UsbTiny	\$125.00	1	\$125.00
Cables	\$100.00	2	\$200.00
Placa experimental	\$170.00	1	\$170.00
Modulo Bluetooth HC-05	\$250.00	1	\$250.00
Fuente de PC	\$400	1	\$400
			\$2,195.00

Tabla 5: Componentes electrónicos y costos

Componentes mecánicos

	Costo	Cantidad	Total
Motor DC	\$150	2	\$300.00
Motor DC con reductor	\$115.00	1	\$115.00
Servo S3003	\$260.00	1	\$260.00
Servo SG90	\$80.00	1	\$80.00
Ruedas	\$150	2	\$300
Tubos Pvc	\$100	1	\$100
Extras	\$150	1	\$300
			\$1,455.00

Tabla 6: Componentes mecánicos y costos

Anexo 4: Código

```
1  .include "m2560def.inc"
2
3  .def IN_COMMAND = r25
4
5  .org 0
6  rjmp initial_config
7
8  .org $0032                //USART0 Rx Complete
9  jmp USART0_RXC
10
11 .org $0064
12 jmp TIM5_OVF              //Timer5 Overflow Handler
13
14
15 .org INT_VECTORS_SIZE
16
17 .include "../Configs/motorsConfiguration.asm"
18 .include "../Configs/usartConfiguration.asm"
19 .include "../Utility/utility.asm"
20 .include "../communicationController.asm"
21 .include "../fireController.asm"
22 .include "../Utility/commandParser.asm"
23 .include "../manualMode.asm"
24 .include "../automaticMode.asm"
25
26 initial_config:
27
28     // Configuro la comunicacion serial
29     call usart_init
30
31     // Configuro motores
32     call motors_init
33     call set_topspin
34
35     ldi r16,(1<<SE)        // Habilito el modo sleep, tipo: IDLE
36     out SMCR, r16
37
38 stand_by:
39     //SETEO ESTADO STAND BY
40     //////////////////////////////////
41     call stop_mixer
42
43     ldi PARAMETER, '0'
44     call set_ball_speed
45
46     ldi PARAMETER, '3'      // 90 grados
47     call set_fire_angle
48
49     //////////////////////////////////
50     //////////////////////////////////
51
52 await_command:
53
54
55     call usart_recieve_command
56     call parse_execute_command
57
58     jmp await_command
```

```
1 .dseg
2 CURRENT_SPIN_TYPE: .byte 1
3
4
5 .cseg
6
7 ldi YL, low(CURRENT_SPIN_TYPE)
8 ldi yH, high(CURRENT_SPIN_TYPE)
9
10 .equ DEFAULT_FIRE_SPEED = 0
11
12 .equ DEFAULT_FIRE_ANGLE = 9 // Angulo 0
13 .equ FIRE_ANGLE_REG = OCR1AL
14
15 .equ DEFAULT_FIRE_STATE = 9 // Angulo 0
16 .equ FIRE_ENABLE_REG = OCR1BL
17
18 .equ FIRE_SPEED_REG1 = OCR3CL // PIN 3
19 .equ FIRE_SPEED_REG2 = OCR3AL // PIN 5
20
21 .equ DEFAULT_MIXING_SPEED = 100
22 .equ MIXING_SPEED_REG = OCR0A
23
24 motors_init:
25     // EN RESUMEN:
26     // MOTOR DISPARADOR 1: PIN 2, 1 o 0 setea direccion de rotacion
27     // MOTOR DISPARADOR 1: PIN 3, PWM
28
29     // MOTOR DISPARADOR 2: PIN 4, 1 o 0 setea direccion de rotacion
30     // MOTOR DISPARADOR 2: PIN 5, PWM
31
32     // MOTOR MEZCLADOR: PIN 13, PWM, el otro pin ponerlo a GND
33
34     // SERVO DIRECCION: PIN 11, PWM, el otro pin ponerlo a GND
35
36     // SERVO ALIMENTADOR: PIN 12 PWM, el otro pin ponerlo a GND
37
38     ldi r16, TOPSPIN
39     st Y, r16 // Seteo el tipo de spin inicial
40
41 pin_config:
42     // Pongo los pines de los puertos necesarios como salida
43     // Uno de los pines va a estar en 0 o 1 dependiendo de la direccion y el otro va a tener el PWM
44     sbi DDRE, PE4 // PIN 2, si esta en 0 va en un sentido si esta en 1 va en el otro
45     sbi DDRE, PE5 // PIN 3 PWM motores disparadores
46
47     sbi DDRE, PG5 // PIN 4, si esta en 0 va en un sentido si esta en 1 va en el otro
48     sbi DDRE, PE3 // PIN 5 PWM motores disparadores
49
50     sbi DDRB, PB7 // PIN 13 PWM motor mezclador
51     // Aca no tengo un pin para la direccion ya que no me interesa, es solo un motor para revolver las
    pelotitas
52
53
54     sbi DDRB, PB5 // PIN 11 PWM servo
55
56     sbi DDRB, PB6 // PIN 12 PWM servo habilitador
57
58 fire_motors_config:
59     lds r16, TCCR3B
60     ori r16, 0b00001001 // WGM32 = 1 CS30 = 1: No prescaling
61     sts TCCR3B, r16
62
63     // Configuro PWM para los motores del disparador
64     lds r16, TCCR3A
65     ori r16, 0b10001001 // COM3C1 = 1, COM3A1 = 1
66     sts TCCR3A, r16
67
68     ldi r16, DEFAULT_FIRE_SPEED
```



```
69     sts FIRE_SPEED_REG1, r16
70     sts FIRE_SPEED_REG2, r16
71
72 mixing_motor_config:
73
74
75     IN r16, TCCR0A
76     sbr r16, 0b1000011      // WGM00 = 1, WGM01 = 1, COM0A1 = 1 ----> PWM, TOP=MAX, Actualiza OCR en ✓
77     TOP
78     OUT TCCR0A, r16
79
80     IN r16, TCCR0B
81     sbr r16, 0b00000001     // CS01 = 1: No prescaling
82     OUT TCCR0B, r16
83
84     ldi r16, DEFAULT_MIXING_SPEED // Velocidad inicial
85     OUT MIXING_SPEED_REG, r16    // Comparador
86
87 fire_enable_motor_config:
88     // WGMn3:0 = 15 en fast pwm ----> OCRnA for defining TOP
89     // Configuro PWM para el servo
90     lds r16, TCCR1A
91     ori r16, 0b00100001      // WGM10 = 1, COM1B1 = 1 ----> PWM, TOP=MAX o MAX CUSTOM, Actualiza OCR ✓
92     en TOP
93     sts TCCR1A, r16
94
95     //El servo requiere de una frecuencia especifica de 50 Hz para funcionar
96     lds r16, TCCR1B
97     ori r16, 0b00001101      // WGM32 = 1, WGM33 = 1, CS30 = 1: prescaler en 256 para lograr una ✓
98     frecuencia de 60 Hz
99     sts TCCR1B, r16
100
101     ldi r16, DEFAULT_FIRE_STATE // Angulo inicial
102     sts FIRE_ENABLE_REG, r16
103
104 servo_motor_config:
105     // WGMn3:0 = 15 en fast pwm ----> OCRnA for defining TOP
106     // Configuro PWM para el servo
107     lds r16, TCCR1A
108     ori r16, 0b10000001      // WGM10 = 1, WGM11 = 1, COM1A1 = 1 COM1A0 = 0 ----> PWM, TOP=MAX o MAX ✓
109     CUSTOM, Actualiza OCR en TOP
110     sts TCCR1A, r16
111
112     //El servo requiere de una frecuencia especifica de 50 Hz para funcionar
113     lds r16, TCCR1B
114     ori r16, 0b00001101      // WGM12 = 1, WGM13 = 1, CS12 = 0, CS11 = 1, CS10 = 1: prescaler en 256 ✓
115     para lograr una frecuencia de 60 Hz
116     sts TCCR1B, r16
117
118     ldi r16, DEFAULT_FIRE_ANGLE // Angulo inicial
119     sts FIRE_ANGLE_REG, r16
120
121     ret
```

```
1
2 .equ BAUD_RATE_L = 103
3 .equ BAUD_RATE_H = 0
4
5 usart_init:
6
7     cli                                // deshabilito interrupciones globales mientras inicializo
8
9     ldi r17, BAUD_RATE_H
10    ldi r16, BAUD_RATE_L               // pongo 103 en los registros r17:r16
11
12    sts UBRR0H, r17
13    sts UBRR0L, r16                   // setea el baud rate en 9600
14
15    ldi r16, (1<<RXEN0)|(1<<RXCIE0)
16    sts UCSR0B,r16                   // activa la recepcion y la interrupcion al terminar una recepcion
17
18    ldi r16, (3<<UCSZ00)               // modo asincronico
19    sts UCSR0C,r16                   // Seteo el format de frame: 8 bits de datos, 1 bit de final
20
21
22    sei                                // vuelvo a activar interrupciones globales
23
24
25    ret
26
27
```

```
1
2 parse_execute_command: // La diferencia con el anterior es que a esta rutina la llamo desde el modo manual
3
4     cpi IN_COMMAND, 's'           // STAND BY
5     breq stand_by_jump
6
7     cpi IN_COMMAND, 'a'           // MODO AUTOMATICO
8     breq receive_auto_mode
9
10    cpi IN_COMMAND, 'b'           // BACKSPIN
11    breq set_backspin_call
12
13    cpi IN_COMMAND, 't'           // TOPSPIN
14    breq set_topspin_call
15
16    cpi IN_COMMAND, 'v'           // SET FIRE SPEED
17    breq receive_fire_speed
18
19    cpi IN_COMMAND, 'n'           // SER FIRE ANGLE
20    breq receive_fire_angle
21
22    cpi IN_COMMAND, 'd'
23    breq shoot_jump
24
25    cpi IN_COMMAND, 'm'
26    breq start_mixer_call
27
28    jmp await_command
29
30 receive_fire_speed:
31     call usart_recieve_command
32     mov PARAMETER, IN_COMMAND
33     call set_ball_speed
34     ret
35
36 receive_fire_angle:
37     call usart_recieve_command
38     mov PARAMETER, IN_COMMAND
39     call set_fire_angle
40     ret
41
42 receive_auto_mode:
43     call usart_recieve_command
44     mov PARAMETER, IN_COMMAND
45
46     // Limpio la interrupcion
47     ldi r16, (1 << RXC0)
48     lds r17, UCSR0A
49
50     or r16, r17
51     sts UCSR0A, r16
52
53     mov PARAMETER, IN_COMMAND
54     jmp automatic_mode
55
56
57 set_topspin_call:
58     call set_topspin
59     ret
60
61 stand_by_jump:
62     jmp stand_by
63
64
65 set_backspin_call:
66     call set_backspin
67     ret
68
69 shoot_jump:
```

```
70         call shoot
71         ret
72
73 start_mixer_call:
74         call start_mixer
75         ret
```

```
1 .equ TOPSPIN = 0b00000000
2 .equ BACKSPIN = 0b00000001
3 .def PARAMETER = r16
4 .def PARAMETER2 = r17
5
6 sixteenth_delay:
7     ldi r18, 5
8     ldi r19, 75
9     ldi r20, 191
10    L5: dec r20
11        brne L5
12        dec r19
13        brne L5
14        dec r18
15        brne L5
16        nop
17    ret
18
19
20 delay_timer:                // 1 segundo delay
21     ldi r16, low(49910)
22     ldi r17, high(49910)
23
24     sts TCNT5H, r17
25     sts TCNT5L, r16
26
27     ldi r16, (1 << TOIE5)
28     sts TIMSK5, r16
29
30     ldi r16, 0b00000101 //(1<<CS50) | (1<<CS52) // Prescaler 1024
31     sts TCCR5B, r16
32     sei
33     sleep
34     ret
35
36 half_delay_timer:          // medio segundo delay
37     ldi r16, low(57723)
38     ldi r17, high(57723)
39
40     sts TCNT5H, r17
41     sts TCNT5L, r16
42
43     ldi r16, (1 << TOIE5)
44     sts TIMSK5, r16
45
46     ldi r16, 0b00000101 //(1<<CS50) | (1<<CS52) // Prescaler 1024
47     sts TCCR5B, r16
48     sei
49     sleep
50     ret
51
52 TIM5_OVF:
53     reti
```

```
1 usart_recieve_command:
2     sleep
3     //call parse_execute_command
4     ret
5
6 has_pending_data:           // me pone el carry en 1 si hay datos para leer en el buffer
7
8     lds r17, UCSRA           // cargo en r17 el registro
9     lsl r17                  // el bit 7 de UCSRA me indica si hay datos para leer
10    ret
11
12 usart_recieve_data: // Si llegue hasta aca es porque hay informacion para leer, leo y vuelvo
13     lds IN_COMMAND, UDR0
14     ret
15
16 USART0_RXC:
17
18     call usart_recieve_data
19
20     reti
21
```

```
1 shoot:
2     // Habilito caida
3     ldi r20, 13
4     sts FIRE_ENABLE_REG, r20
5     call sixteenth_delay
6     // deshabilito caida
7     ldi r20, 9
8     sts FIRE_ENABLE_REG, r20
9     ret
10
11
12 set_ball_speed:
13
14     ld r18, Y
15     cpi r18, BACKSPIN
16     brne its_topspin           // Si es backspin tengo que complementar la velocidad
17
18     call map_backspin_firespeed_ToPWM
19     sts FIRE_SPEED_REG1, PARAMETER2
20     sts FIRE_SPEED_REG2, PARAMETER
21     ret
22
23 its_topspin:
24     call map_topspin_firespeed_ToPWM           // Mapeo el valor recibido por parametro 0-5 a un valor de PWM 0-255
25     sts FIRE_SPEED_REG2, PARAMETER2
26     sts FIRE_SPEED_REG1, PARAMETER
27
28     ret
29
30 set_fire_angle:
31     call map_FireAngle_ToPWM
32     sts FIRE_ANGLE_REG, PARAMETER // asumo que en r16 viene el angulo entre 1 y 5
33     ret
34
35 start_mixer:
36     ldi PARAMETER, DEFAULT_MIXING_SPEED
37     OUT MIXING_SPEED_REG, PARAMETER // asumo que el parametro de la velocidad viene en r16
38     ret
39
40 stop_mixer:
41     ldi PARAMETER, 0
42     OUT MIXING_SPEED_REG, PARAMETER // asumo que el parametro de la velocidad viene en r16
43     ret
44
45 set_topspin:
46     ld r17, Y           // Tengo el tipo de spin actual en r17
47     cpi r17, TOPSPIN
48     breq return         // si ya tenia topspin no hago nada
49     call swich_speeds   // y si es distinto tengo que invertir la velocidad(complementar el PWM)
50
51     ldi r17, TOPSPIN
52     st Y, r17           // guardo el tipo de spin
53
54     ret
55
56
57 set_backspin:
58     ld r17, Y           // Tengo el tipo de spin actual en r17
59     cpi r17, BACKSPIN
60     breq return         // si ya tenia backspin no hago nada
61     call swich_speeds   // y si es distinto tengo que invertir la velocidad(complementar el PWM)
62
63     ldi r17, BACKSPIN
64     st Y, r17           // guardo el tipo de spin
65
66     ret
67
```

```
68 return:
69     ret
70
71 swich_speeds:
72
73     lds r1, FIRE_SPEED_REG1
74     lds r2, FIRE_SPEED_REG2
75
76     sts FIRE_SPEED_REG1, r2
77     sts FIRE_SPEED_REG2, r1
78
79     ret
80
81
82 map_topspin_firespeed_ToPWM:
83     cpi PARAMETER, '0'
84     breq set_topspin_fire_to_0
85     cpi PARAMETER, '1'
86     breq set_topspin_fire_to_1
87     cpi PARAMETER, '2'
88     breq set_topspin_fire_to_2
89     cpi PARAMETER, '3'
90     breq set_topspin_fire_to_3
91     cpi PARAMETER, '4'
92     breq set_topspin_fire_to_4
93     cpi PARAMETER, '5'
94     breq set_topspin_fire_to_5
95
96     jmp set_topspin_fire_to_0                // Si el parametro es mayor a 5 o un caracter invalido
97
98 set_topspin_fire_to_0:
99     ldi PARAMETER, 3
100    ldi PARAMETER2, 3
101    ret
102 set_topspin_fire_to_1:
103     ldi PARAMETER, 105
104     ldi PARAMETER2, 75
105     ret
106 set_topspin_fire_to_2:
107     ldi PARAMETER, 115
108     ldi PARAMETER2, 75
109     ret
110 set_topspin_fire_to_3:
111     ldi PARAMETER, 130
112     ldi PARAMETER2, 75
113     ret
114 set_topspin_fire_to_4:
115     ldi PARAMETER, 140
116     ldi PARAMETER2, 75
117     ret
118 set_topspin_fire_to_5:
119     ldi PARAMETER, 155
120     ldi PARAMETER2, 75
121     ret
122
123 map_backspin_firespeed_ToPWM:
124     cpi PARAMETER, '0'
125     breq set_backspin_fire_to_0
126     cpi PARAMETER, '1'
127     breq set_backspin_fire_to_1
128     cpi PARAMETER, '2'
129     breq set_backspin_fire_to_2
130     cpi PARAMETER, '3'
131     breq set_backspin_fire_to_3
132     cpi PARAMETER, '4'
133     breq set_backspin_fire_to_4
134     cpi PARAMETER, '5'
135     breq set_backspin_fire_to_5
136
```



```
137     jmp set_backspin_fire_to_0           // Si el parametro es mayor a 5 o un caracter invalido
138
139     set_backspin_fire_to_0:
140         ldi PARAMETER, 3
141         ldi PARAMETER2, 3
142         ret
143     set_backspin_fire_to_1:
144         ldi PARAMETER, 105
145         ldi PARAMETER2, 70
146         ret
147     set_backspin_fire_to_2:
148         ldi PARAMETER, 110
149         ldi PARAMETER2, 70
150         ret
151     set_backspin_fire_to_3:
152         ldi PARAMETER, 115
153         ldi PARAMETER2, 70
154         ret
155     set_backspin_fire_to_4:
156         ldi PARAMETER, 120
157         ldi PARAMETER2, 70
158         ret
159     set_backspin_fire_to_5:
160         ldi PARAMETER, 125
161         ldi PARAMETER2, 70
162         ret
163
164     map_FireAngle_ToPWM:
165
166         cpi PARAMETER, '1'
167         breq set_angle_to_1
168         cpi PARAMETER, '2'
169         breq set_angle_to_2
170         cpi PARAMETER, '3'
171         breq set_angle_to_3
172         cpi PARAMETER, '4'
173         breq set_angle_to_4
174         cpi PARAMETER, '5'
175         breq set_angle_to_5
176
177
178     jmp set_angle_to_1
179
180     set_angle_to_1:           // 0 GRADOS
181         ldi PARAMETER, 14     // Estos numeros son arbitrarios, salen de probar fisicamente el resultado
182         ret
183     set_angle_to_2:           // 45 GRADOS
184         ldi PARAMETER, 18
185         ret
186     set_angle_to_3:           // 90 GRADOS
187         ldi PARAMETER, 23
188         ret
189     set_angle_to_4:           // 135 GRADOS
190         ldi PARAMETER, 27
191         ret
192     set_angle_to_5:           // 180 GRADOS
193         ldi PARAMETER, 31
194         ret
195
```

```
1
2 automatic_mode:
3
4     cpi PARAMETER, '1'
5     breq auto_mode1
6
7     cpi PARAMETER, '2'
8     breq auto_mode2_jump
9
10    cpi PARAMETER, '3'
11    breq auto_mode3_jump
12
13    ldi IN_COMMAND, 0
14
15    jmp stand_by
16
17 auto_mode3_jump:
18     jmp auto_mode3
19
20 auto_mode2_jump:
21     jmp auto_mode2
22
23 auto_mode1: // Aca vario la velocidad efecto y ubicacion de la pelota para generar ejercicios
    preestablecidos
24     call start_mixer
25     call set_topspin
26
27     // BOLA 1
28     ldi PARAMETER, '5'
29     call set_ball_speed
30     ldi PARAMETER, '5'
31     call set_fire_angle
32
33     // DELAY ANTES DE EMPEZAR
34     call delay_timer
35     call delay_timer
36     call delay_timer
37     call delay_timer
38     ///
39
40     call shoot
41
42     call delay_timer
43     call half_delay_timer
44
45     // BOLA 2
46     ldi PARAMETER, '3'
47     call set_fire_angle
48
49     call delay_timer
50
51     call shoot
52
53     call delay_timer
54     call half_delay_timer
55
56     // BOLA 3
57     ldi PARAMETER, '5'
58     call set_fire_angle
59
60     call delay_timer
61
62     call shoot
63
64     call delay_timer
65     call half_delay_timer
66
67
68     // BOLA 4
```

```
69     ldi PARAMETER, '1'
70     call set_fire_angle
71
72     call delay_timer
73
74     call shoot
75
76     cpi IN_COMMAND, 's'
77     breq go_to_standby
78
79     jmp auto_mode1
80
81
82 auto_mode2:
83
84     call start_mixer
85
86     call set_topspin
87
88     // PARA QUE ARRANQUE MAS RAPIDO
89     ldi PARAMETER, '5'
90     call set_ball_speed
91     ///
92
93     call delay_timer
94     call delay_timer
95
96     // BOLA 1
97     ldi PARAMETER, '3'
98     call set_ball_speed
99     ldi PARAMETER, '5'
100    call set_fire_angle
101
102    // DELAY ANTES DE EMPEZAR
103
104    call delay_timer
105    call delay_timer
106    call delay_timer
107    ///
108
109    call shoot
110
111    // BOLA 2
112    ldi PARAMETER, '5'
113    call set_ball_speed
114
115    call delay_timer
116    call shoot
117
118    // BOLA 3
119    ldi PARAMETER, '5'
120    call set_ball_speed
121
122    call delay_timer
123    call shoot
124
125    cpi IN_COMMAND, 's'
126    breq go_to_standby
127
128    jmp auto_mode2
129
130 go_to_standby:
131     jmp stand_by
132
133 auto_mode3:
134     call start_mixer
135
136     call set_topspin
137
```

```
138      // PARA QUE ARRANQUE MAS RAPIDO
139      ldi PARAMETER, '5'
140      call set_ball_speed
141      ///
142
143      call delay_timer
144      call delay_timer
145
146      // BOLA 1
147      ldi PARAMETER, '3'
148      call set_ball_speed
149      ldi PARAMETER, '1'
150      call set_fire_angle
151
152      // DELAY ANTES DE EMPEZAR
153
154      call delay_timer
155      call delay_timer
156      call delay_timer
157      ///
158
159      call shoot
160
161      // BOLA 2
162      ldi PARAMETER, '5'
163      call set_ball_speed
164
165      call delay_timer
166      call shoot
167
168      // BOLA 3
169      ldi PARAMETER, '5'
170      call set_ball_speed
171
172      call delay_timer
173      call shoot
174
175      cpi IN_COMMAND, 's'
176      breq go_to_standby
177
178      jmp auto_mode3
179
180
```