Taller de proyecto II

**Bitácora Proyecto 5**

Aguilar Sergio - Ailigo Oriana

línea horizontal

# 

# 

## 10/9/2018

Se comenzó con la lectura de la continuación del proyecto del coche anticolisión, logramos ponernos en contexto con los objetivos del proyecto y conseguimos informarnos acerca de las características del proyecto (implementación, componentes, documentación).

## 13/9/2018

Se comenzó con la investigación de motores de CC y sistemas de control de lazo cerrado,  
y se documento referencias respecto al mismo. El artículo leido explica cómo usar el módulo de controlador de motor de doble canal HG7881 (L9110).  
  
El HG7881 (L9110) es un chip de controlador de motor compacto que admite un rango de voltaje de 2.5-12 V a 800 mA de corriente continua. Estos chips tienen diodos de abrazadera de salida incorporados para proteger su electrónica sensible del microcontrolador. Son adecuados para proyectos de robots muy pequeños.  
  
Cada chip HG7881 (L9110) puede conducir un solo motor de CC con dos entradas de control digitales. Una entrada se usa para seleccionar la dirección del motor mientras que la otra se usa para controlar la velocidad del motor. La velocidad se controla utilizando la modulación de ancho de pulso PWM. Los conductores de motor suelen tener lo que se llama una tabla de verdad que determina el efecto de sus entradas. La tabla de verdad para un solo chip HG7881 (L9110) es la siguiente:

**HG7881 (L9110) tabla de verdad del motor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **entrada** | | **salida** | | |
| **IA** | **IB** | **OA** | **IB** | **estado del motor** |
| **L(bajo)** | **L** | **L** | **L** | **apagado** |
| **H(alto)** | **L** | **H** | **L** | **Adelante** |
| **L** | **H** | **L** | **H** | **Marcha atrás** |
| **H** | **H** | **H** | **H** | **apagado** |

Tenga en cuenta que la dirección real de "adelante" e "atrás" depende de cómo se montan y cablean los motores. Siempre puede cambiar la dirección de un motor invirtiendo su cableado.  
  
El módulo de controlador de motor de doble canal HG7881 (L9110) utiliza dos de estos chips de controlador de motor. Cada chip del controlador está destinado a conducir un motor, por lo que tener dos significa que este módulo puede controlar dos motores de forma independiente. Cada canal de motor usa la misma tabla de verdad que la anterior. Cada juego de terminales de tornillo se utiliza para conectar un motor. Consulte la tabla a continuación para conocer las conexiones de los cabezales de los pines.

HG7881 (L9110) Conector de módulo de controlador de motor de doble canal

|  |  |
| --- | --- |
| **Pin** | **Descripcion** |
| B-IA | Motor B entrada A (IA) |
| B-IB | Motor B salida B (IB) |
| GND | Ground |
| VCC | Tensión de funcionamiento 2.5-12V |
| A-IA | Motor A entrada A (IA) |
| A-IB | Motor A entrada (IB) |

Recomendamos usar la entrada 1A para controlar la velocidad de cada motor y la entrada 1B para controlar la dirección.

**Conexiones:**  
   
 Salida digital Arduino D10 a la entrada del controlador del motor B-IA.  
 Salida digital Arduino D11 a la entrada del controlador del motor B-IB.  
 Motor controlador VCC a voltaje de operación 5V.  
 Motor conductor GND a tierra común.  
 Motor del conductor MOTOR B atornille los terminales a un motor pequeño  
  
  
El siguiente boceto de Arduino muestra cómo controlar un solo motor:

// conexiones cableadas

#define HG7881\_B\_IA 5 // D10 -> Motor B Entrada A -> MOTOR B +

#define HG7881\_B\_IB 4 // D11 -> Motor B Entrada B -> MOTOR B -

// conexiones funcionales

#define MOTOR\_B\_PWM HG7881\_B\_IA // Velocidad PWM del motor B

#define MOTOR\_B\_DIR HG7881\_B\_IB // Motor B Dirección

// los valores reales para "rápido" y "lento" dependen del motor

#define PWM\_SLOW 50 // ciclo de trabajo arbitrario de baja velocidad PWM

#define PWM\_FAST 200 // ciclo de trabajo arbitrario de velocidad rápida PWM

#define DIR\_DELAY 1000 // breve retraso para cambios bruscos de motor

void setup()

{

Serial.begin( 9600 );

pinMode( MOTOR\_B\_DIR, OUTPUT );

pinMode( MOTOR\_B\_PWM, OUTPUT );

digitalWrite( MOTOR\_B\_DIR, LOW );

digitalWrite( MOTOR\_B\_PWM, LOW );

}

void loop()

{

boolean isValidInput;

// dibujar un menú en el puerto serie

Serial.println( "-----------------------------" );

Serial.println( "MENU:" );

Serial.println( "1) Avance rápido" );

Serial.println( "2) Adelante" );

Serial.println( "3) parar suave (detenerse despacio)" );

Serial.println( "4) marcha atras" );

Serial.println( "5) marcha atras rapido" );

Serial.println( "6) Parada de golpe (freno)" );

Serial.println( "-----------------------------" );

do

{

byte c;

// obtener el siguiente carácter del puerto serie

Serial.print ("?");

while (! Serial.available ())

; // LAZO...

c = Serial.read ();

// ejecuta la opción de menú según el personaje recibido

switch( c )

{

case '1': // 1) Avance rápido

Serial.println ("Avance rápido ...");

// siempre detiene los motores brevemente antes de cambios abruptos

digitalWrite (MOTOR\_B\_DIR, LOW);

digitalWrite (MOTOR\_B\_PWM, LOW);

delay (DIR\_DELAY);

// establece la velocidad y dirección del motor

digitalWrite (MOTOR\_B\_DIR, HIGH); // direccion = adelante

analogWrite (MOTOR\_B\_PWM, 255-PWM\_FAST); // velocidad PWM = rápido

isValidInput = true;

break;

case '2': // 2) adelante

Serial.println( "adelante..." );

// siempre detenga los motores brevemente antes de cambios abruptos

digitalWrite( MOTOR\_B\_DIR, LOW );

digitalWrite( MOTOR\_B\_PWM, LOW );

delay( DIR\_DELAY );

// establecer la velocidad y dirección del motor

digitalWrite( MOTOR\_B\_DIR, HIGH ); // direccion = adelante

analogWrite( MOTOR\_B\_PWM, 255-PWM\_SLOW ); // velocidad del PWM = lento

isValidInput = true;

break;

case '3': // 3) Parada suave (preferida)

Serial.println( "Tope suave (costa) ..." );

digitalWrite( MOTOR\_B\_DIR, LOW );

digitalWrite( MOTOR\_B\_PWM, LOW );

isValidInput = true;

break;

case '4': // 4) Reversa

Serial.println( "Avance rápido..." );

// siempre detenga los motores brevemente antes de cambios abruptos

digitalWrite( MOTOR\_B\_DIR, LOW );

digitalWrite( MOTOR\_B\_PWM, LOW );

delay( DIR\_DELAY );

//establecer la velocidad y dirección del motor

digitalWrite( MOTOR\_B\_DIR, LOW ); // direccion = reversa

analogWrite( MOTOR\_B\_PWM, PWM\_SLOW ); // velocidad del PWM = lento

isValidInput = true;

break;

case '5': // 5) reverso rapido

Serial.println( "adelante rapido..." );

// siempre detenga los motores brevemente antes de cambios abruptos

digitalWrite( MOTOR\_B\_DIR, LOW );

digitalWrite( MOTOR\_B\_PWM, LOW );

delay( DIR\_DELAY );

// establecer la velocidad y dirección del motor

digitalWrite( MOTOR\_B\_DIR, LOW ); // direccion = reversa

analogWrite( MOTOR\_B\_PWM, PWM\_FAST ); // velocidad del PWM = rapido

isValidInput = true;

break;

case '6': // 6) Parada de golpe (uso con precaución)

Serial.println( "parada de golpe (frenar)..." );

digitalWrite( MOTOR\_B\_DIR, HIGH );

digitalWrite( MOTOR\_B\_PWM, HIGH );

isValidInput = true;

break;

default:

// caracter invalido! vea el menu otra vez!

isValidInput = false;

break;

}

} while( isValidInput == true );

// repite el main loop y dibuja el menu...

}

/\*EOF\*/

**HG7881\_Motor\_Driver\_Example - Arduino sketch**

Es importante tener en cuenta que el módulo de controlador de motor de doble canal HG7881 (L9110) es muy simple. El voltaje utilizado para conducir el chip es el mismo voltaje utilizado para conducir los motores. Esto significa que si está utilizando un chip de 5V, debe esperar conducir motores de 5V. Elija un controlador de motor de mayor rendimiento para obtener más opciones de potencia y control.

**Enlaces de referencias para tener en cuenta respecto a los motores:**

Video de control de velocidad PWM ANÁLISIS y CONTROL:

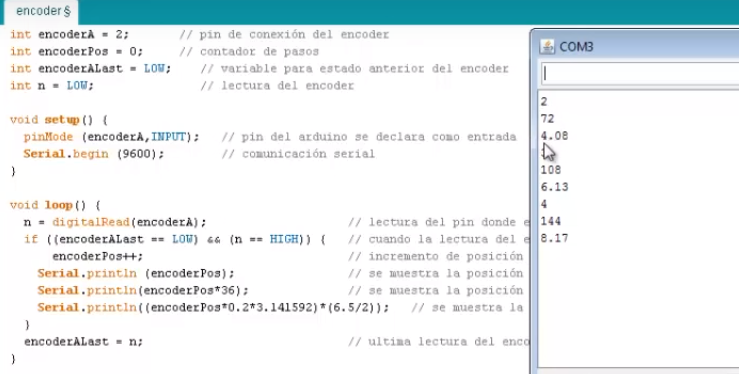
<https://www.youtube.com/watch?v=BMncYdbufng>

Video de control de motores:

<https://www.youtube.com/watch?v=mVo8DfEsRno>

**Encoder**

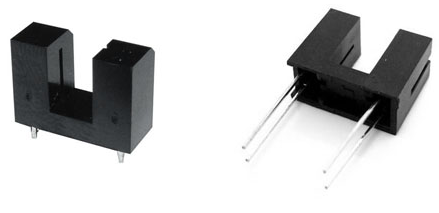
**Probamos el funcionamiento del encoder viendo el siguiente video:** [**https://www.youtube.com/watch?v=t9Zy8UvJXsI**](https://www.youtube.com/watch?v=t9Zy8UvJXsI)

****

**Figura 1: Primera prueba del encoder**

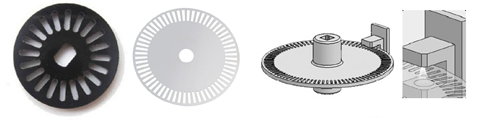
# **¿QUÉ ES UN OPTOINTERRUPTOR?**

**Un optointerruptor es un sensor con forma de “U” que permite detectar un objeto que atraviesa el dispositivo por la ranura interior.**

****

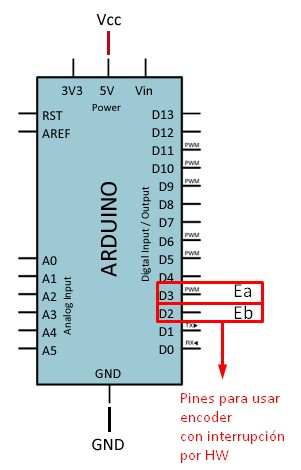
**Los optointerruptores son sensores sencillos. Uno de los extremos contiene un diodo emisor de infrarrojos, mientras que el otro contiene un fototransistor que recibe la señal. Cuando un objeto pasa por la ranura interrumpe el rayo de luz infrarroja, lo que es detectado por el fototransistor.**

**Los optointerruptores son muy empleados como encoder para detectar la velocidad de giro y posición del eje de un motor. Para ello se emplea un disco con ranuras acoplado al eje.**

****

# **ESQUEMA ELÉCTRICO**

**Alimentamos el módulo a través de Vcc y GND conectándolos, respectivamente, a la salida de 5V y GND en Arduino.**



# **EJEMPLOS DE CÓDIGO**

Tenemos varias opciones para leer un optointerruptor con Arduino. Si estamos detectando la presencia de un objeto, simplemente leemos el estado de la entrada digital, tal y como vimos en la entrada [Entradas digitales en Arduino](https://www.luisllamas.es/entradas-digitales-en-arduino/).

Cuando el sensor se dispara, ejecutamos las acciones necesarias, cómo incrementar un contador, o medir el tiempo entre disparos.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | const int sensorPin = 9;    void setup() {  Serial.begin(9600); //iniciar puerto serie  pinMode(sensorPin , INPUT); //definir pin como entrada  }  void loop(){  int value = 0;  value = digitalRead(sensorPin ); //lectura digital de pin  if (value == LOW) {  Serial.println("Optointerruptor activado");  }  delay(1000);  } |

Sin embargo, en el caso de usar el optointerruptor como encoder lo normal es que empleemos las interrupciones de Arduino, lo que nos simplificará considerablemente el código. El punto negativo es que tendremos que hacer debounce (tiempo de rebote) a las entradas y que Arduino UNO (se dispone) y Nano (no lo estamos usando) solo tenemos dos interrupciones externas, lo cuál en algunos vehículos se nos quedará corto.

# **QUÉ SON Y CÓMO USAR INTERRUPCIONES EN ARDUINO**

Si queremos detectar un cambio de estado en esta entrada, el método que hemos usado hasta ahora es emplear las entradas digitales para consultar repetidamente el valor de la entrada, con un intervalo de tiempo (delay) entre consultas.

Este mecanismo se denomina “poll”, y tiene 3 claras desventajas:

* Suponer un continuo consumo de procesador y de energía, al tener que preguntar continuamente por el estado de la entrada.
* Si la acción necesita ser atendida inmediatamente, por ejemplo en una alerta de colisión, esperar hasta el punto de programa donde se realiza la consulta puede ser inaceptable.
* Si el pulso es muy corto, o si el procesador está ocupado haciendo otra tarea mientras se produce, es posible que nos saltemos el disparo y nunca lleguemos a verlo.

Interrupción: Es un mecanismo que permite asociar una función a la ocurrencia de un determinado evento. Cuando ocurre el evento el procesador “sale” inmediatamente del flujo normal del programa y ejecuta la función ISR asociada ignorando por completo cualquier otra tarea (por esto se llama interrupción). Al finalizar la función ISR asociada, el procesador vuelve al flujo principal, en el mismo punto donde había sido interrumpido.

Nota: Para usar interrupciones en dispositivos físicos (como pulsadores, sensores ópticos, … ) debemos antes eliminar el efecto “rebote”, como podéis ver en la entrada [Aplicar debounce al usar interrupciones en Arduino](https://www.luisllamas.es/debounce-interrupciones-arduino/). En nuestro caso, eliminamos ese efecto rebote mediante hardware ya que colocamos las resistencias para que cuando haya valores de voltajes 0,3 V por ejemplo, se dirija directamente hacia GND.

**INTERRUPCIONES EN ARDUINO**  
Arduino dispone de dos tipos de eventos en los que definir interrupciones. Por un lado tenemos las interrupciones de timers y por otro lado, tenemos las interrupciones de hardware, que responden a eventos ocurridos en ciertos pines físicos.

Dentro de las interrupciones de hardware, que son las que nos ocupan en esta entrada, Arduino es capaz de detectar los siguientes eventos.

* RISING, ocurre en el flanco de subida de LOW a HIGH.
* FALLING, ocurre en el flanco de bajada de HIGH a LOW.
* CHANGING, ocurre cuando el pin cambia de estado (rising + falling).
* LOW, se ejecuta continuamente mientras está en estado LOW.

Los pines susceptibles de generar interrupciones varían en función del modelo de Arduino.

En Arduino y Nano se dispone de dos interrupciones, 0 y 1, asociados a los pines digitales 2 y 3.El Arduino Mega dispone de 6 interrupciones, en los pines 2, 3, 21, 20, 19 y 18 respectivamente. Arduino Due dispone de interrupciones en todos sus pines.



## LA FUNCIÓN ISR

La función asociada a una interrupción se denomina ISR (Interruption Service Routines) y, por definición, tiene que ser una función que no recibe nada y no devuelva nada.

Dos ISR no pueden ejecutarse de forma simultánea. En caso de dispararse otra interrupción mientras se ejecuta una ISR, la función ISR se ejecuta una a continuación de otra.

Al diseñar una ISR debemos mantener como objetivo que tenga el menor tiempo de ejecución posible, dado que mientras se esté ejecutando el bucle principal y todo el resto de funciones se encuentran detenidas.

Imaginemos, por ejemplo, que el programa principal ha sido interrumpido mientras un motor acercaba un brazo para coger un objeto. Una interrupción larga podría hacer que el brazo no para a tiempo, tirando o dañando el objeto.

Frecuentemente la función de la ISR se limitará a activar un flag, incrementar un contador, o modificar una variable. Esta modificación será atendida posteriormente en el hilo principal, cuando sea oportuno.

Nota: No empleéis en una ISR un proceso que consuma tiempo. Esto incluye cálculos complejos, comunicación (serial, I2C y SPI) y, en la medida de lo posible, cambio de entradas o salidas tanto digitales como analógicas.

#### **LAS VARIABLES DE LA ISR COMO “VOLATILES”**

Para poder modificar una variable externa a la ISR dentro de la misma debemos declararla como “volatile”. El indicador “volatile” indica al compilador que la variable tiene que ser consultada siempre antes de ser usada, dado que puede haber sido modificada de forma ajena al flujo normal del programa (lo que, precisamente, hace una interrupción).

Al indicar una variable como Volatile el compilador desactiva ciertas optimizaciones, lo que supone una pérdida de eficiencia. Por tanto, sólo debemos marcar como volatile las variables que realmente lo requieran, es decir, las que se usan tanto en el bucle principal como dentro de la ISR.

#### **EFECTOS FUERA DE LA ISR**

Durante la ejecución de una interrupción Arduino no actualiza el valor de la función millis y micros. Es decir, el tiempo de ejecución de la ISR no se contabiliza y Arduino tiene un desfase en la medición del tiempo.

Si un programa tiene muchas interrupciones y estas suponen un alto tiempo de ejecución, la medida del tiempo de Arduino puede quedar muy distorsionada respecto a la realidad (nuevamente, un motivo para hacer las ISR cortas).

#### **EFECTOS DENTRO DE LA ISR**

Dentro de la ISR el resto de interrupciones están desactivadas. Esto supone:

* La función millis no actualiza su valor, por lo que no podemos utilizarla para medir el tiempo dentro de la ISR. (sí podemos usarla para medir el tiempo entre dos ISR distintas)
* Como consecuencia la función delay() no funciona, ya que basa su funcionamiento en la función millis()
* La función micros() actualiza su valor dentro de una ISR, pero empieza a dar mediciones de tiempo inexactas pasado el rango de 500us.
* En consecuencia, la función delayMicroseconds funciona en ese rango de tiempo, aunque debemos evitar su uso porque no deberíamos introducir esperas dentro de una ISR.

## CREAR INTERRUPCIONES EN ARDUINO

Para definir una interrupción en Arduino usamos la función:

|  |  |
| --- | --- |
| 1 | attachInterrupt(interrupt, ISR, mode); |

Donde interrupt es el número de la interrupción que estamos definiendo, ISR la función de callback asociada, y mode una de las opciones disponibles (Falling, Rising, Change y Low)

No obstante, es más limpio emplear la función digitalPinToInterrupt(), que convierte un Pin a la interrupción equivalente. De esta forma se favorece el cambio de modelo de placa sin tener que modificar el código.

|  |  |
| --- | --- |
| 1 | attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); |

Otras funcionas interesantes para la gestión de interrupciones son:

* DetachInterrupt(interrupt ), anula la interrupción.
* NoInterrupts(), desactiva la ejecución de interrupciones hasta nueva orden. Equivale a sli()
* Interrupts(), reactiva las interrupciones. Equivale a cli()

### ¿Qué es PID?

"Un controlador PID calcula un valor de 'error' como la diferencia entre una [Entrada] medida y un punto de ajuste deseado. El controlador intenta minimizar el error ajustando [una Salida]".

Entonces, le dice al PID qué debe medir (la "Entrada",) Dónde quiere que esté esa medición (el "Punto de ajuste",) y la variable para ajustar que puede hacer que eso suceda (la "Salida".) El PID entonces ajusta la salida tratando de hacer que la entrada sea igual al punto de ajuste.

Para referencia, en un automóvil, la entrada, el punto de ajuste y la salida serían la velocidad, la velocidad deseada y el ángulo del acelerador respectivamente.

### Parámetros de ajuste

Cuando hablamos de CÓMO ajusta la salida para dirigir la entrada hacia el punto de ajuste. Hay 3 Parámetros de Sintonización (o "Afinaciones"): Kp, Ki y Kd. Ajustar estos valores cambiará la forma en que se ajusta la salida. ¿Rápido? ¿Lento? Todo esto se puede lograr dependiendo de los valores de Kp, Ki y Kd.

Entonces, ¿cuáles son los valores de ajuste "correctos" para usar? No hay una respuesta correcta. Los valores que funcionan para una aplicación pueden no funcionar para otra, al igual que el estilo de conducción que funciona para un camión puede no funcionar para un auto de carrera. Con cada nueva aplicación, deberá probar varios valores de Afinación hasta que encuentre un conjunto que le ofrezca lo que desea.

Nota: Hay una [biblioteca de autoajuste PID](https://playground.arduino.cc/Code/PIDAutotuneLibrary) que puede determina los parámetros de ajuste.

Cuando el auto se mueva, y a veces, acelera y, a veces, reduzca la velocidad. Esto no es repetible, por lo tanto, no es necesario utilizar el PID. Pero en nuestro caso sería una buena idea utilizarlo, ya que estariamos dandole un valor constante para la velocidad.

Funciones:

[PID ()](https://playground.arduino.cc/Code/PIDLibraryConstructor)

[Calcular()](https://playground.arduino.cc/Code/PIDLibraryCompute)

[Modo de ajuste()](https://playground.arduino.cc/Code/PIDLibrarySetMode)

[SetOutputLimits ()](https://playground.arduino.cc/Code/PIDLibrarySetOutputLimits)

[SetTunings ()](https://playground.arduino.cc/Code/PIDLibrarySetTunings)

[SetSampleTime ()](https://playground.arduino.cc/Code/PIDLibrarySetSampleTime)

[SetControllerDirection ()](https://playground.arduino.cc/Code/PIDLibrarySetControllerDirection)

[Funciones de visualización](https://playground.arduino.cc/Code/PIDLibraryDisplayFunctions)

## ALGO SOBRE MOTORES DE C.C. (CORRIENTE CONTINUA)

El motor CC. es una máquina encargada de convertir la energía eléctrica en energía mecánica.

Estos motores están formados por dos partes fundamentalmete:

**-Estator:** Es la parte inmóvil del motor y esta compuesto por unos potentes imanes.

**-Rotor:** Es la parte móvil del motor. Esta parte está compuesta por una serie de bobinas de hilos de cobre enrrolladas en un núcleo ferreomagnético.

## CONCEPTO DE RELACIÓN DE REDUCCIÓN EN UN MOTORREDUCTOR

Se necesita que cuando gire la rueda “A” a su vez también gire la rueda “B” pero sucederá que por cada tres vueltas que dé “A”, la rueda “B” solamente dará una vuelta, esto es, el diámetro de “B” dividido por el diámetro de “A” (15/5 = 3). Este número 3 será la relación de reducción de este reductor o motorreductor elemental y se indica como 3:1

Con esta simple combinación se ha logrado disminuir la velocidad de rotación de la rueda “B” a la tercera parte de la velocidad de la rueda “A”. Con este sistema de reducción no solamente disminuimos la velocidad de “B” a un giro más lento que es útil para la mayoría de las aplicaciones sino que al mismo tiempo estaremos aumentado el “par” o “torque” en la última rueda del motorreductor que generalmente se conoce como la rueda de salida a la que va ensamblada la “flecha de salida” del reductor o motorreductor.

## 22/9/2018

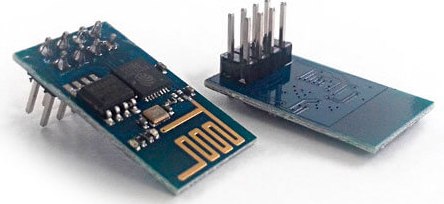
se realizó la investigación de la configuración del módulo ESP8266-WIFI (esquema, comandos , configuración) , queda pendiente la prueba de conexión.

# **¿QUÉ ES EL ESP8266?**

El ESP8266 es un microprocesador de bajo coste con Wifi integrado fabricado por Espressif. Podemos usar el ESP8266 para conectar nuestros proyectos de electrónica y robótica con Arduino. En realidad, el ESP8266 es mucho más que un módulo Wifi para Arduino. Es un procesador completo, con mucha más potencia que la mayoría de modelos de Arduino.

Por otro lado, en cuanto a comunicación Wifi, el ESP01 tiene comunicación integrada 802.11 b/g/n, incluidos modos Wi-Fi Direct (P2P) y soft-Ap. Incluye una pila de TCP/IP completa, lo que libera de la mayor parte del trabajo de comunicación al procesador.

Podemos usar el ESP01 para dotar de conectividad WiFi a nuestros proyectos con Arduino. Sin embargo, como comentamos con los [módulos de Ethernet](https://www.luisllamas.es/tag/ethernet/), la comunicación con Internet puede suponer una carga excesiva para Arduino.

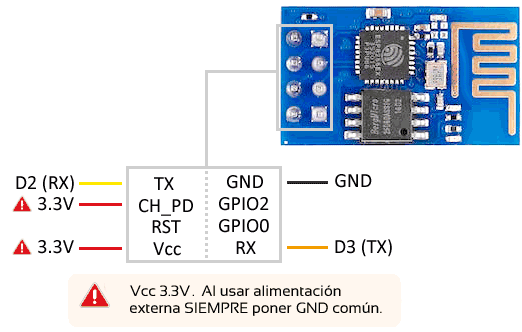


# **ESQUEMA DE MONTAJE**

La principal dificultad es la alimentación del ESP01. El ESP8266 y, en particular, el ESP01, tiene una tensión de alimentación de 3.3V. En ningún caso puede alimentarse a una tensión superior a 3.6V, o dañaremos el módulo.

Por otro lado, el consumo del módulo puede sobrepasar los 200mA, sobre todo durante la conexión y arranques. Sin embargo, el regulador de voltaje de 3.3V de Arduino sólo puede proporcionar 50mA (150mA en algunos modelos), lo cual es insuficiente para alimentar el ESP01.

Por tanto, necesitaremos alimentar el ESP01 con una fuente externa de 3.3V. De lo contrario experimentaremos continuos cortes y cuelgues durante su funcionamiento, que además pueden reducir la vida útil del ESP01.



El resto de la conexión no tiene ninguna dificultad. Por un lado tenemos el pin CH\_PD que apaga o enciende el módulo conectándolo, respectivamente, a Gnd o 3.3V. Por su parte, el pin RST reinicia el módulo si se conecta a Gnd. En algunas versiones del módulo podremos dejarlo sin conexión pero, en general, tendremos que conectarlo a 3.3V para que el módulo arranque.

Finalmente, la comunicación con el módulo se realiza mediante puerto serie. A diferencia de la alimentación del módulo, que en ningún caso debe exceder los 3.6V, existe un amplio debate sobre si los pines RX y TX son tolerantes a 5V (Es decir, si pueden conectarse directamente a un sistema a un 5V).

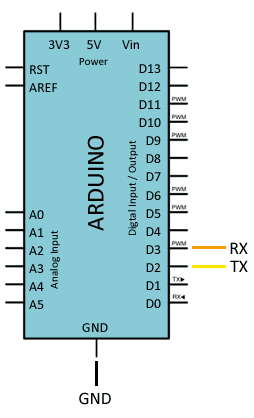
En principio, conectar RX y TX a un sistema de 5V no parece dañar la placa de forma inmediata. De hecho, encontramos indicios en la documentación que apuntan a que podrían ser tolerantes a 5V, sin quedar totalmente claro. No obstante, tampoco queda claro si puede reducir la vida útil del componente.

Para la conexión con Arduino podemos emplear el puerto serie normal, pero esto implica que tendremos que desconectar el ESP01 cuando queramos programar Arduino, y no podremos usar el puerto serie para la comunicación con el PC.

Por este motivo frecuentemente se emplea un puerto serie por software. No obstante, tener en cuenta que esto supone una carga de procesado importante para Arduino.

En los esquemas de esta entratal 2 y RX en el pin digital 3, pero podríamos definir el prto n cualquiera otro pin I/O.

Por tanto, la conexión vista desde Arduino sería la siguiente.



En cuanto a las velocidades, el ESP01 puede configurarse a 9600, 19200, 38400, 74880, 115200, 230400, 460800 y 921600. Por defecto suelen venir, según fabricante, a 9600 o 115200.

## USO DEL ESP8266 CON COMANDOS AT

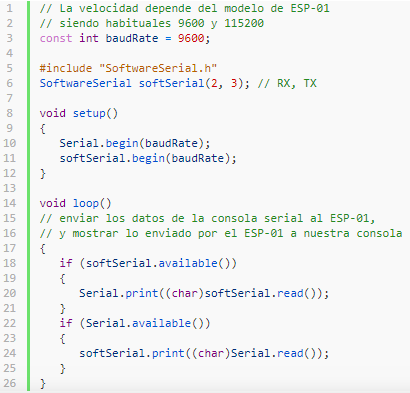
La comunicación con el ESP01 con el firmware por defecto se realiza a través de comandos AT, que son comandos de texto enviados por Serial.

Podemos enviar estos comandos por un conversor USB-TTL (FT232, CH340G o CP2102) o, en nuestro caso, usando Arduino y Software serial como adaptador.

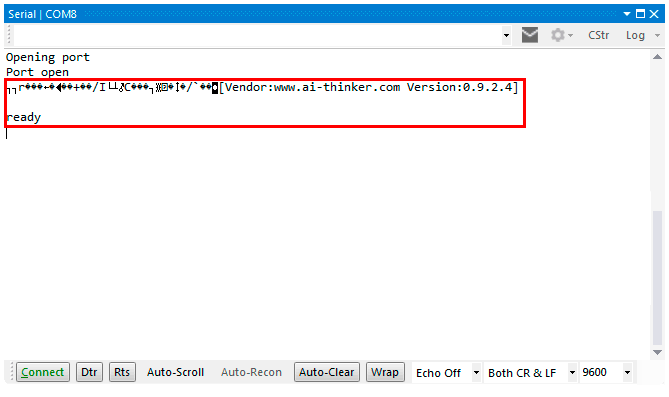
### PRIMERA PRUEBA

Vamos a hacer la primera prueba de conexión con el ESP01. Para ello conectamos el ESP01 a Arduino como hemos visto en el apartado anterior. Dejamos conectado Arduino al ordenador por USB.

A continuación cargamos el siguiente Sketch en Arduino, que seguramente alguno reconocerá como el programa Serial Loop. Este sketch únicamente actúa “puente” entre el puerto serie hardware conectado con el PC, y el puerto serie Soft conectado al ESP01.



Una vez cargado el Sketch, encendemos (o reiniciamos) el ESP01. En el Monitor Serie el ESP01 responde con una serie de caracteres que dependen del fabricante y modelo, y finalmente “Ready”, indicando que el módulo está listo.



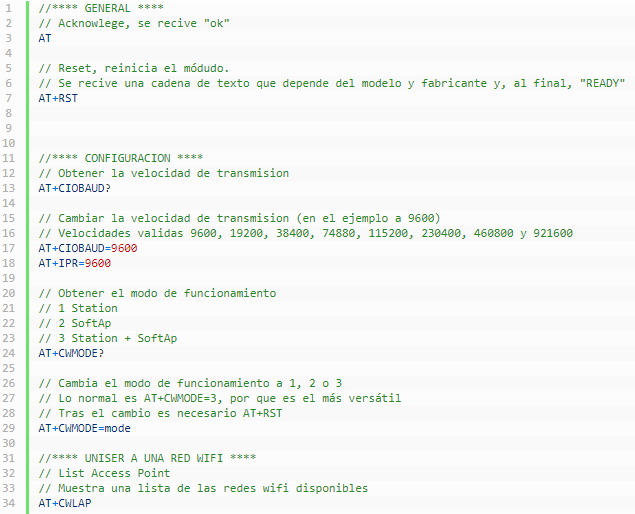
Si ahora escribimos AT, el módulo responderá con “OK”, indicando de nuevo que el módulo está listo para su uso.

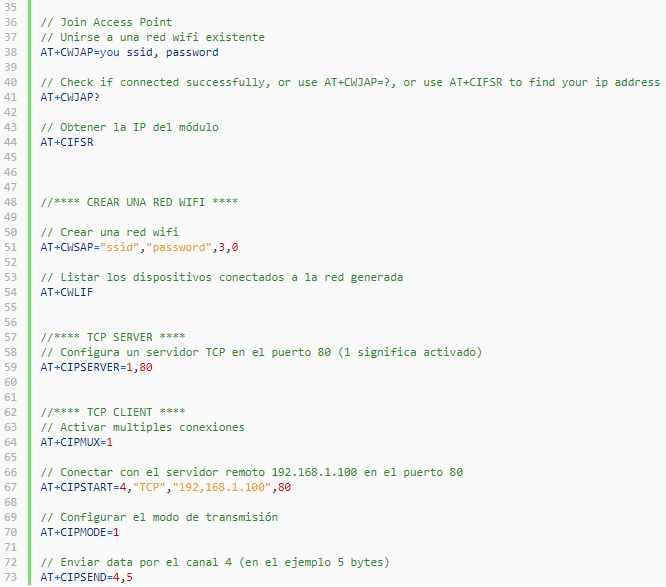


Si no se ve el mensaje inicial finalizado en “Ready” y en su lugar aparecen “caracteres raros”, se debe cambiar la velocidad del puerto Serie en el Sketch Serial Loop, y en el Monitor Serie.

### LISTADO COMANDOS AT

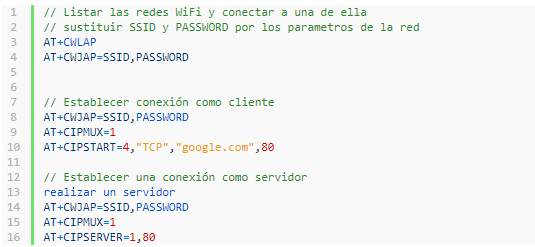
A continuación, un listado de algunos de los comandos AT disponibles para el ESP8266.





### EJEMPLOS AT

Aquí tenemos algunos ejemplos sencillos para realizar acciones básicas con el ESP01 a través de comandos AT.



Por ejemplo, para configurar el ESP01 desde un Arduino para actuar como servidor pasaríamos los comandos AT de la siguiente forma.



## 22/9/2018

Se realizaron pruebas de funcionamiento de los motores, mediante la interfaz serie del arduino. Se descubrieron problemas en la rotación de los 2 motores en un determinado sentido de giro (antihorario).  
NOTA: el funcionamiento del giro es relativamente bueno cuando se sigue la siguiente secuencia :

1)marcha atrás   
 2)marcha hacia adelante velocidad 2  
3)marcha hacia adelante velocidad 1

Por otro lado, queda pendiente terminar de mejorar el funcionamiento de las ruedas y realizar configuraciones con respecto al encoder.

## 26/9/2018

Se realizaron pruebas de funcionamiento del módulo wifi y surgieron errores a la hora de ingresar comandos debido a la forma en que se enviaba el string   
se consiguió establecer el módulo como acces point, pero falla el comando AT para darle nombre y contraseña, se continúa investigando acerca de este inconveniente.

## 

## 

## 26/9/2018

Se realizaron pruebas de funcionamiento del módulo wifi y surgieron errores a la hora de ingresar comandos debido a la forma en que se enviaba el string.  
Por otro lado, se consiguió establecer el módulo como acces point, pero falla el comando AT para darle nombre y contraseña, se continúa investigando acerca de este inconveniente.

# **02/10/2018**

# Solo nos concentramos en el módulo wifi junto con el arduino 1. Por lo tanto, se procedió con el comando AT para darle nombre y contraseña. Después de muchos intentos, se logró hacerlo. Nos dimos cuenta que se deben escribir todos los parámetros:

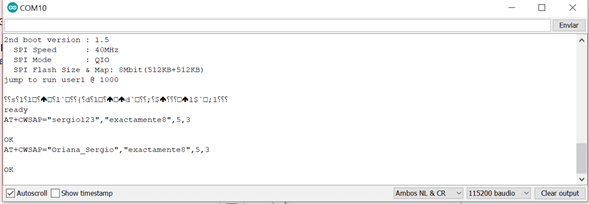
# **-Oriana\_Sergio**: Es el nombre del host.

# **-exactamente8**: Es la contraseña que tiene como máximo una capacidad de 64 bytes.

# **-5**: Es el id del canal.

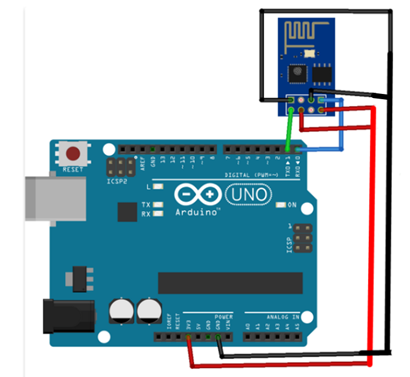
# **-3**: Es el modo de seguridad (WPA2\_PSK).

# y que la contraseña a establecer debe tener como mínimo 8 caracteres, como se observa en la figura 1:



# Figura 1: Cambiar nombre y contraseña del módulo wifi.

# Las pruebas se realizaron mediante esta conexión:



# Figura 2: esquema de conexionado.

# Nota: Tener en cuenta que se mandan comandos a la terminal, y que la comunicación del arduino se realiza mediante los pines 0 (RX) y 1 (TX).

# 

# 

# Figura 3: Conexionado físico Módulo ESP8266 y Arduino Uno

# 

**06/10/2018**

Para ver en detalle cada componente se realizó otro tipo de coneccion. Por lo tanto, se compró lo siguiente con un total de **515 $** :

* 45 Pines macho-macho: 170 $ (ver figura 4)

- 30 pines hembra-macho: 120 $ (ver figura 4)



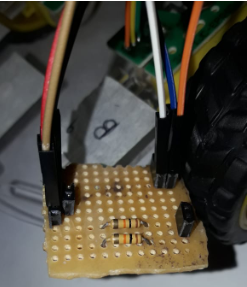
# figura 4: Pines macho-macho y pines hembra-macho

- 100 gramos de estaño: 200 $ (ver figura 5)



# figura 5: Estaño 60/40 plomo

- Placa: 25 $

****

# **figura 6: PCB para conexión de encoder’s**

## 7/10/2018

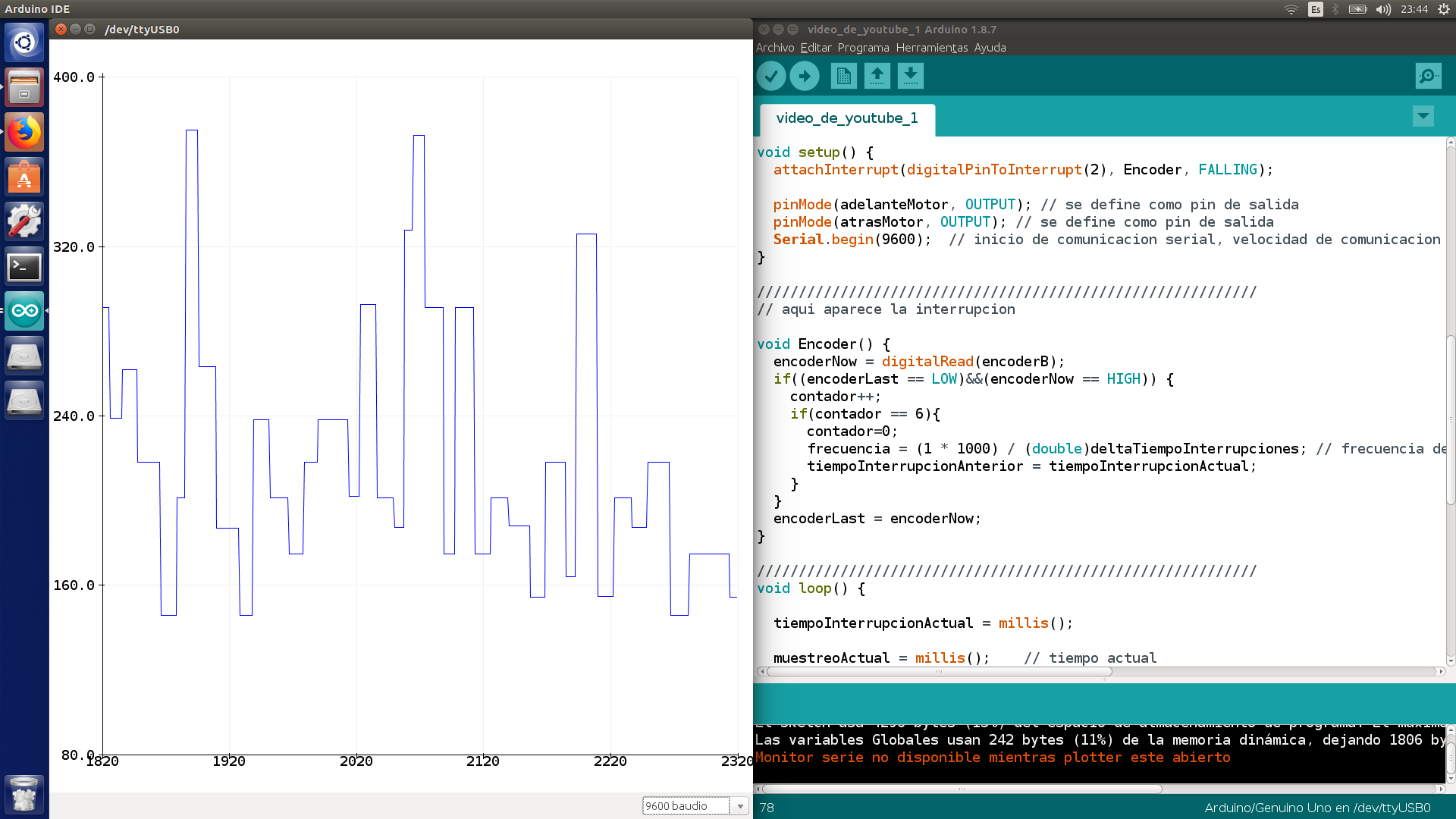
Se realizaron pruebas de medición de rpm para una rueda del autito. Para esto se implementó el manejo de interrupciones de hardware mediante un encoder que dispara eventos. Cada vez que se activa un evento se realiza un cálculo de frecuencia relativo a cada cuanto una rueda realiza una vuelta completa. El código para el cálculo es el siguiente:

void Encoder() {

|  |
| --- |
| encoderNow = digitalRead(encoderB);  if((encoderLast == LOW)&&(encoderNow == HIGH)) {  contador++;  if(contador == 6){  contador=0;  frecuencia = (1 \* 1000) / (double) deltaTiempoInterrupciones; // frecuencia de las interrupciones en Hz  tiempoInterrupcionAnterior = tiempoInterrupcionActual;  }  }  encoderLast = encoderNow; } |

para establecer el valor de contador (variable de control de una vuelta) se pensó en lo siguiente:

“ como el disco de encoder posee 6 ranuras se estima que una vuelta de la rueda del autito equivale a 6 transiciones entre estados LOW y HIGH “



# **figura 7: medición de velocidad de una rueda del autito**

8/10/2018

Realizando pruebas con el encoder, se salio y se rompio el mismo, ambas ruedas. Luego el optointerruptor de una de las ruedas se quemo.

21/10/18

Con respecto a la velocidad de cada rueda, se logró llegar a un valor en común para ambas. En la siguiente figura se ve el valor al que converge la rotación de los dos motores. Este valor será elegido como valor deseado para el algoritmo del PID para el control de la velocidad de los motores.

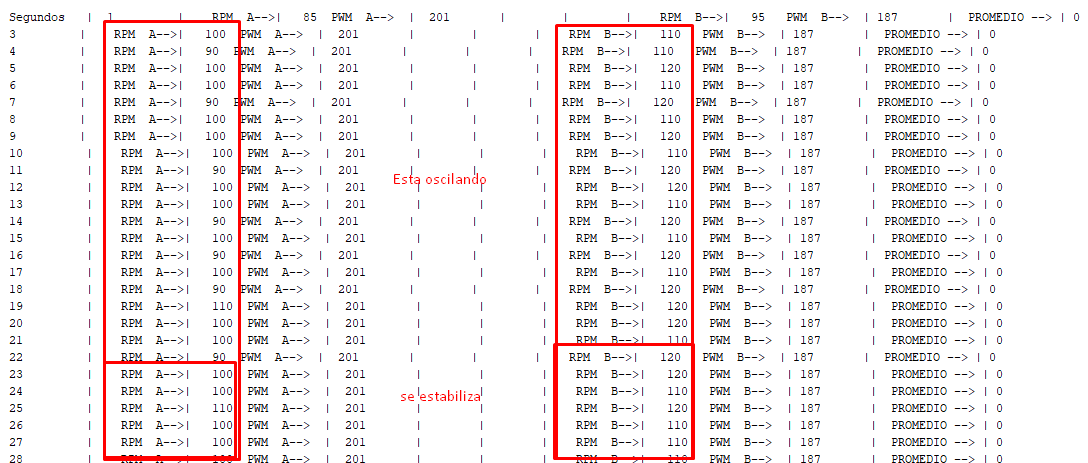
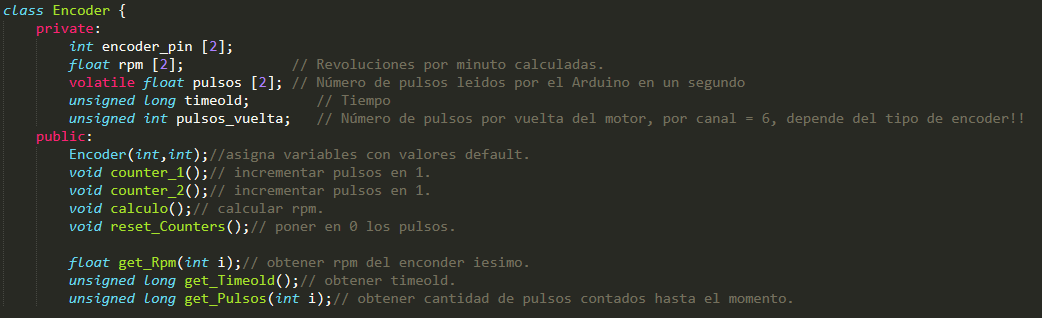


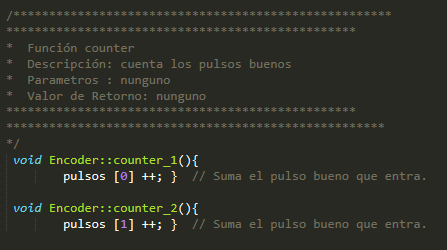
Figura 8: Valor de RPM deseado.

# 24/10/2018

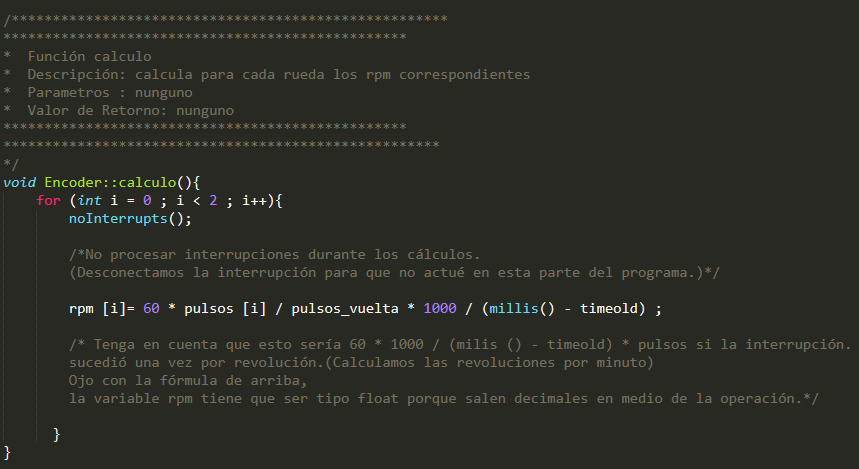
24/10/2018

Se implementó un librería para la medición de la velocidad llamada Encoder.h, con las siguientes funciones: 

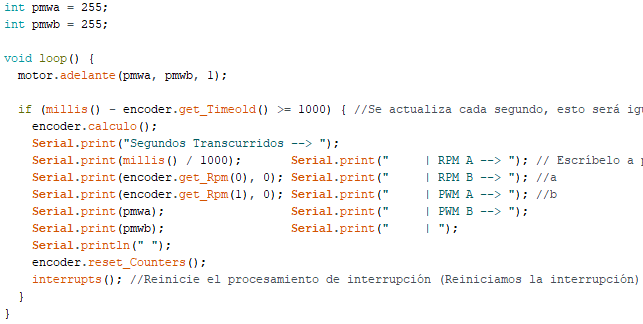
Función que cuenta el número de pulsos este es llamado como rutina de servicio para atender una interrupción:



Función que realiza el cálculo de RPM para cada rueda:



con el siguiente algoritmo podemos ver la velocidad y el PWM aplicado a los motores A y B



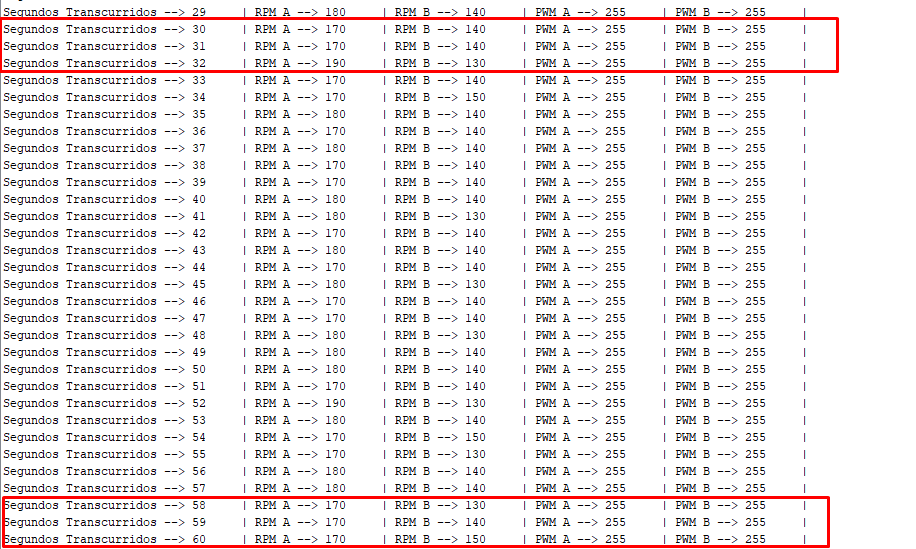


Figura 8: variación de RPM a partir de los 60 segundos y los 30 segundos

27/10/2018

Se implementó un librería para el control de motores llamada L9110.h, con las siguientes funciones:

|  |
| --- |
| class L9110 {  private:  int \_aia;  int \_aib;  int \_bia;  int \_bib;  public:  L9110(int aia,int aib,int bia,int bib);  void adelante(int i,int d,int t);  void atras(int i,int d,int t);  void izquierda(int i,int d,int t);  void derecha(int i, int d,int t);  void frenar(int t); }; |

Las siguientes son funciones para el desplazamiento del carrito:

|  |
| --- |
| void L9110::adelante(int i,int d,int t){//desplazamiento hacia adelante  digitalWrite(\_aib,LOW);  digitalWrite(\_bib,LOW);  analogWrite(\_aia,i);  analogWrite(\_bia,d);  delay(t); }  void L9110::atras(int i,int d,int t){//desplazamiento hacia atras  digitalWrite(\_aib,LOW);  digitalWrite(\_bib,LOW);  analogWrite(\_aia,i);  analogWrite(\_bia,d);   delay(t); }  void L9110::izquierda(int i,int d,int t){//desplazamiento hacia izquier  analogWrite(\_aia,i);  digitalWrite(\_aib,LOW);   digitalWrite(\_bia,LOW);  analogWrite(\_bib,d);  delay(t); }  void L9110::derecha(int i,int d,int t){//desplazamiento hacia derecha  digitalWrite(\_aia,0);  analogWrite(\_bia,d);  analogWrite(\_aib,i);  digitalWrite(\_bib,0);  delay(t); }  void L9110::frenar(int t){//para total del desplazamiento  analogWrite(\_aia,0);  analogWrite(\_bia,0);  digitalWrite(\_aib,0);  digitalWrite(\_bib,0);  delay(t); } |

la funcionalidad adelante() fue usada en la implementación del código anterior

En cada función se pueden apreciar 3 parámetros a excepción de frenar que solo tiene tiempo, para los demás los primero 2 parámetros representan el pwm a aplicar al motor 1 y 2 respectivamente.

24/10/2018

Se Utilizó librería PID\_V1 para conseguir que el auto ande recto con las siguientes funciones:

|  |
| --- |
| PID(double\*, double\*, double\*, double, double, double, int); //constructor. vincula el PID a la entrada, la salida y el punto de ajuste. Los parámetros iniciales de ajuste también se establecen aquí   void SetMode(int Mode);// \* establece PID en Manual (0) o Automático (no 0)  bool Compute(); // realiza el cálculo PID. debería llamarse cada ciclo de tiempo () ciclos. ON / OFF y la frecuencia de cálculo se pueden configurar usando Set Mode, Set SampleTime respectivamente  void SetOutputLimits(double, double); //sujeta la salida a un rango específico. 0-255 por defecto, pero es probable que el usuario quiera cambiar esto dependiendo de la aplicación |
|  |

COMUNICACIÓN CON EL PID:

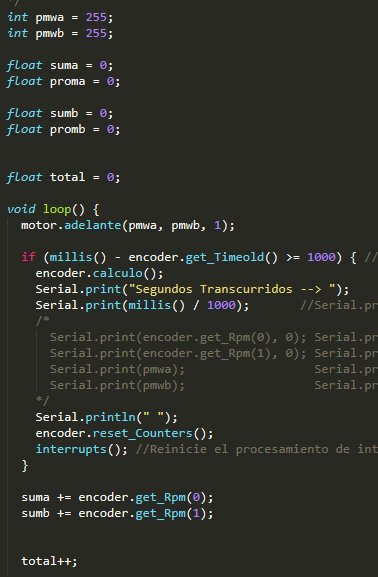
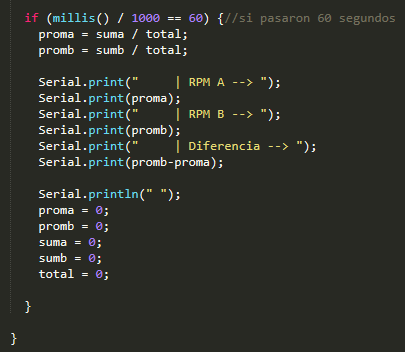
En el constructor se establece como parámetros Input, Output y Setpoint las cuales serían las variables de control para el funcionamiento de los motores.

|  |
| --- |
| * La acción de control aplicada se obtiene a través de del segundo parámetro del constructor * La señal de entrada se envia por referencia a través del primer parámetro * El valor de ajuste deseado se envía a través del tercer parámetro |

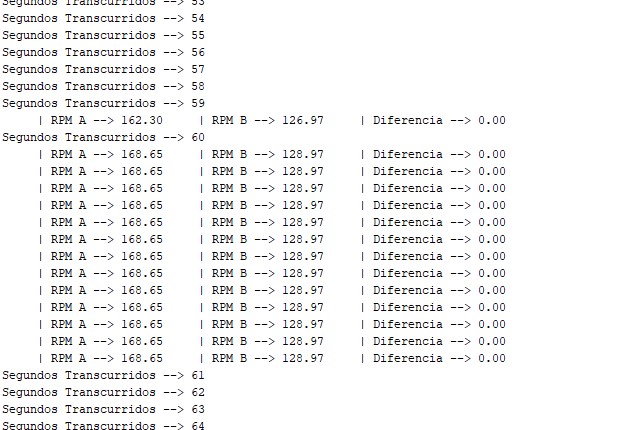
25/10/18

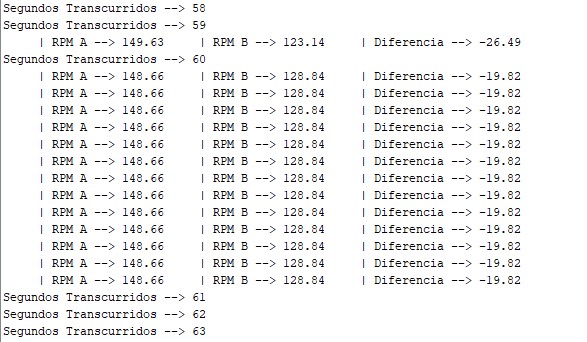
## PROMEDIO DE RPM DE AMBOS MOTORES Y LA DIFERENCIA ENTRE ELLOS

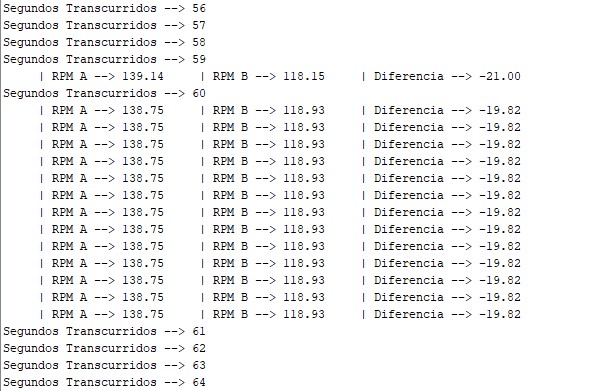
Con el siguiente código se tomaron 4 muestras del promedio de velocidad de rotación que se logra obtener a partir de aplicar un ciclo de trabajo de 255, y además la diferencia promedio de velocidad rotación entre una rueda y otra.

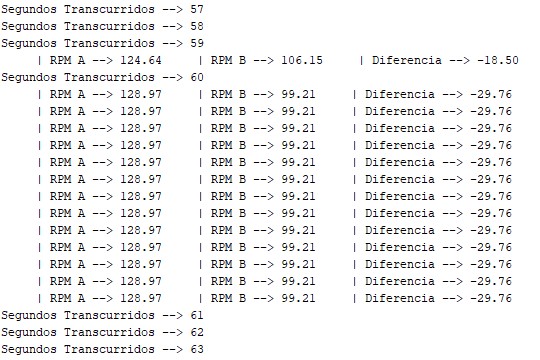


## 4 muestras









**En las primeras 2 muestras :**

|  |  |
| --- | --- |
| RPM A | **148.66 -+ 20.02** |
| RPM B | **128.84 -+ 0.13** |
| DIFERENCIA | **19.82** |

**En las últimas 2 muestras con la batería un poco descargada(luego de 40 min de uso):**

|  |  |
| --- | --- |
| RPM A | **128.97 -+ 9.78** |
| RPM B | **99.21 -+ 19.72** |
| DIFERENCIA | **29.76 -+ 9.94** |

**En ambos casos la rueda A, presenta mayor velocidad de giro para un ciclo de trabajo de 255**

**la diferencia de velocidad en las ruedas escala según el consumo a razón de -+10 rpm**

**según la carga en la batería que alimenta al controlador Puente H L9110**

27/10/18

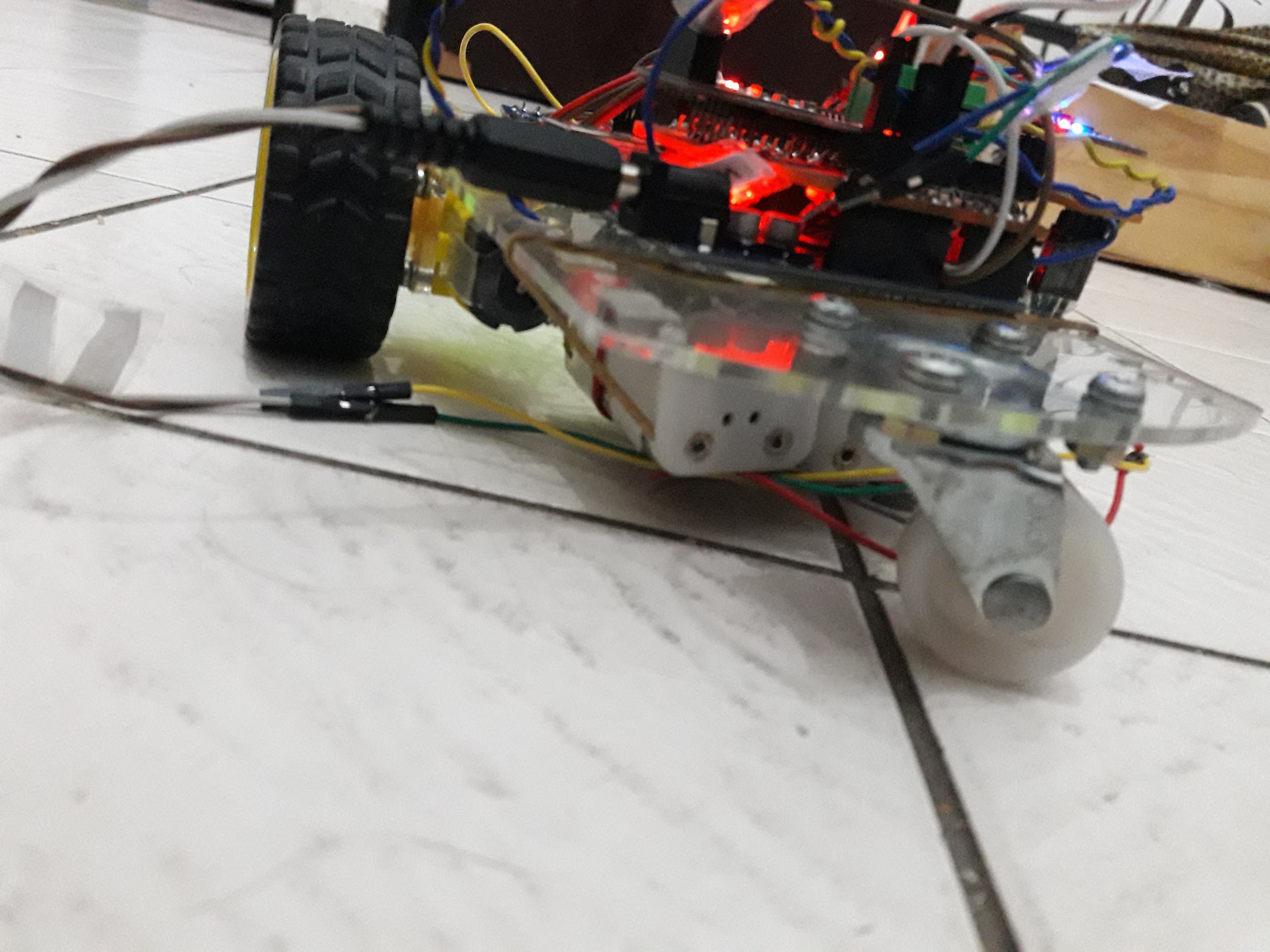
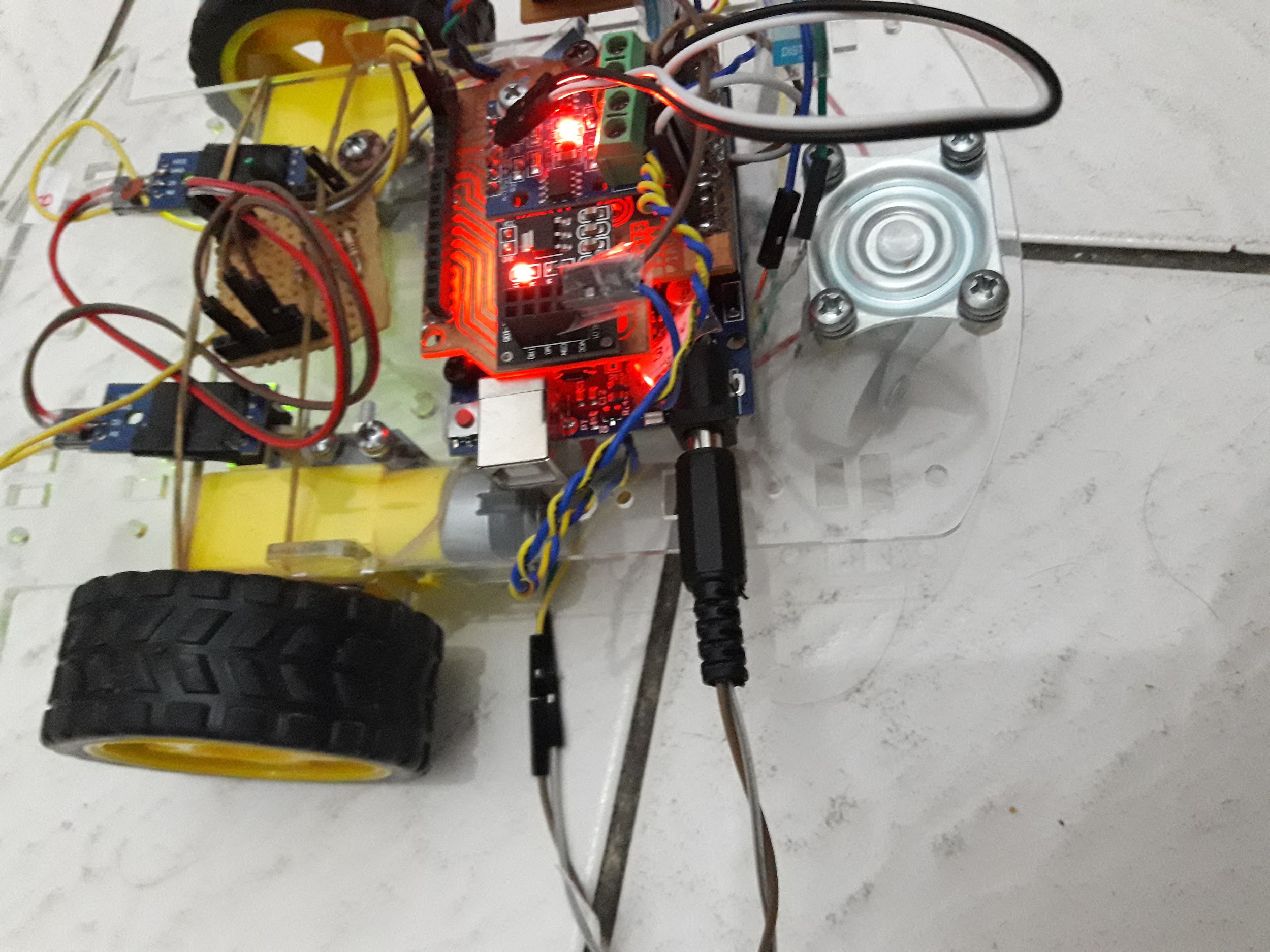
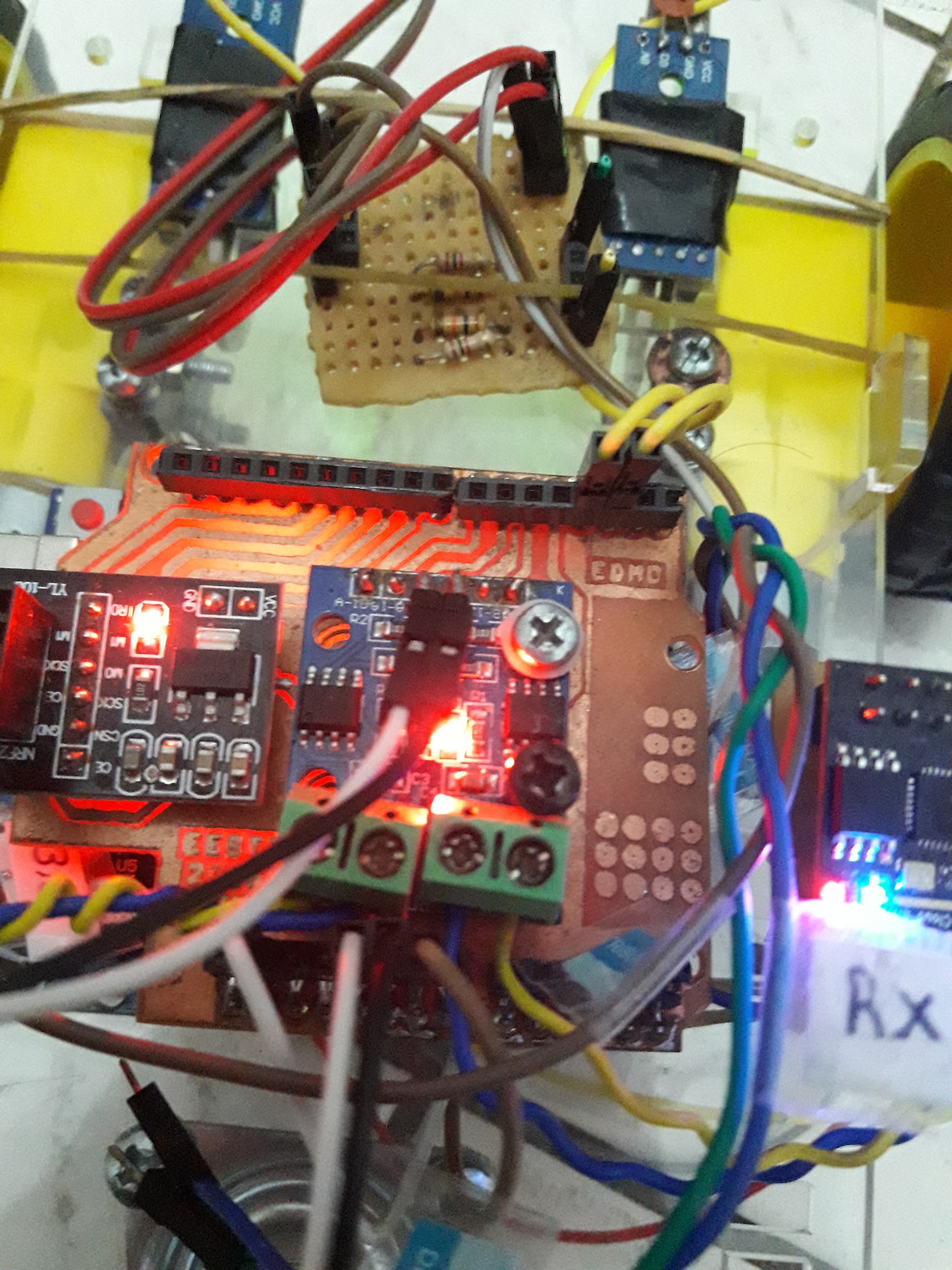
para corroborar la medición no basamos de la implementación del mismo en base a la siguiente fuente la cual nos explica lo siguiente acerca de los rpm de un tacómetro:

|  |
| --- |
| “Un tacómetro tiene un sensor de pulso magnético al lado de un engranaje de hierro en un eje de motor. El engranaje tiene 30 dientes. La lectura del pulso indica 90 pulsos por segundo. ¿Cuál es el RPM del eje del motor? El RPM = pulsos por segundo multiplicado por 60 segundos por minuto dividido por el número de dientes del engranaje. El RPM del eje del motor = 90 multiplicado por 60 dividido entre 30 = 180 RPM” |

<https://www.puromotores.com/13119985/cual-es-la-fomula-para-calcular-las-revoluciones-por-minuto-o-rpm>

28/10/18

Logramos que las dos ruedas se estabilice en un valor de rpm de referencia fijado por nosotros. Pero todavía posee una leve desviación, ya que ambas ruedas trabajan con un pid distinto y los tiempo de estabilización de cada una no es la misma (teniendo en cuenta perturbación del peso de la batería el cual produce un ligero desvío y fuerza al motor a trabajar con el máximo de carga que le brinda la batería. Por lo tanto, un motor no puede generar más velocidad ya que el peso de las pilas se lo impide. Otro factor es el tipo de superficie por donde el vehículo se desplaza, aunque este no influye tanto). Por otro lado, intentamos realizar la prueba sin pilas y solo usando alimentación del puerto serial ,y otra variante de alimentación para arduino a través del jack power de alimentación mediante la cual se alimenta todo el arduino. Estos intentos también fueron un fracaso, ya que influye la perturbación externa producida por el cable mismo. Como conclusión no importa cual sea la carga de alimentación sobre el auto (teniendo como valor límite 11v suministrados al puente h y 7.5v al arduino uno por el jack ambas cargas son independientes), siempre tendrá un leve desvío.



CODIGO UTILIZADO PARA LAS PRUEBAS REALIZADAS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Proyecto 5

Medición de encoders

Autores: Ailigo Oriana , Aguilar Sergio

Fecha: 11/10/2018

para mas informacion acerca del uso de la biblioteca PID --> URL: https://playground.arduino.cc/Code/PIDLibrary

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include<L9110.h>

#include<Encoder.h>

#include <PID\_v1.h>

/\*VARIABLES FLOAT PARA PID\*/

//double Kp = 0.5, Ki = 0.16, Kd = 0.15;

double Kp = 0.5, Ki = 0.10, Kd = 0.09; // no se toca

double Kpb = 0.5, Kib = 0.12, Kdb = 0;

double rpmA, rpmB, Setpointa, Setpointb, Outputa , Outputb;

/\*HASTA ACA VARIABLES FLOAT\*/

//Objets

L9110 motor(6, 7, 5, 4);//Este es el orden de los motores A y B respectivamente

Encoder encoder(2, 3);//encoder A = 2 , encoder B = 3

PID myPID(&rpmA, &Outputa, &Setpointa, Kp, Ki, Kd, DIRECT);

PID myPIDb(&rpmB, &Outputb, &Setpointb, Kpb, Kib, Kdb, DIRECT);

//Fin de Objets

//RUTINA DE SERVICIO DE INTERRUPCION (ISR)

void contar1() {

encoder.counter\_1();

}

void contar2() {

encoder.counter\_2();

}

//FIN de (ISR)

void setup() {

Serial.begin(9600); // Configuración del puerto serie

attachInterrupt(0, contar1, FALLING); // La interrupcion 0 es el pin digital 2, de modo que es donde se conecta el detector IR. Disparadores en FALLING (cambio de ALTO a BAJO) // Configuración de la interrupción 0, donde esta conectado el encoder A.

attachInterrupt(1, contar2, FALLING); // La interrupcion 1 es el pin digital 3, de modo que es donde se conecta el detector IR. Disparadores en FALLING (cambio de ALTO a BAJO) // Configuración de la interrupción 1, donde esta conectado el encoder B.

Setpointa = 90; // velocidad referencia

Setpointb = 90; // velocidad referencia

//turn the PID on

myPID.SetMode(AUTOMATIC); // MANUAL = pid apagado ,AUTOMATIC = pid activado

myPIDb.SetMode(AUTOMATIC);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Programa Principal de Arduino UNO

Descripción: XXXXXXXXXX

Parametros : nunguno

Valor de Retorno: nunguno

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

//int pwma = 135;

//int pwmb = 150;

int pwma = 150;

int pwmb = 150;

void loop() {

motor.adelante(pwma, pwmb, 1);

if (millis() - encoder.get\_Timeold() >= 1000) { //Se actualiza cada segundo, esto será igual a la frecuencia de lectura (Hz).

// Serial.print(" | pulsos A --> "); Serial.print(encoder.get\_Pulsos(0)); Serial.print(" | pulsos B --> "); Serial.print(encoder.get\_Pulsos(1));

encoder.calculo();

interrupts(); //Reinicie el procesamiento de interrupción (Reiniciamos la interrupción)

/\*IMPRESION EN MONITOR SERIAL DE LOS RPMS Y LOS PWM\*/

// Serial.print("Segundos Transcurridos --> "); Serial.print(millis() / 1000);

Serial.print(" | RPM A --> "); Serial.print(encoder.get\_Rpm(0), 0); Serial.print(" | RPM B --> "); Serial.print(encoder.get\_Rpm(1), 0); //Serial.print(" | Diferencia --> "); Serial.print(encoder.get\_Rpm(1) - encoder.get\_Rpm(0), 0);

Serial.print(" | PWM A --> "); Serial.print(pwma); Serial.print(" | PWM B --> "); Serial.print(pwmb); Serial.print(" |Output --> "); Serial.print(Outputa); Serial.print(" |Outputbb --> "); Serial.print(Outputb);

Serial.println(" ");

}

rpmA = encoder.get\_Rpm(0);

rpmB = encoder.get\_Rpm(1);

myPID.Compute();

myPIDb.Compute();

if ((encoder.get\_Rpm(1) == 0) && (encoder.get\_Rpm(0) == 0 )) {

pwma = Outputa;

pwmb = Outputb;

}

else {

if (encoder.get\_Rpm(1) - encoder.get\_Rpm(0) != 0) {

pwma = Outputa;

pwmb = Outputb;

}

}

}

29/10/18

Realizando pruebas en el piso, notamos que las constantes que teníamos antes, no nos servían, ya que las mismas se usaban cuando el auto no tocaba el suelo. Por lo tanto, debemos encontrarlas nuevamente, comenzando con un Kp nuevo.

-kp=1.5 con valor deseado 80 rpm

podemos observar que kp multiplica el valor deseado obteniendo una salida de 120 pwm. Este valor de salida impide que el auto se mueva.

-kp=1.65 con valor deseado 80 rpm

entonces la salida obtenida es 130 pwm . Este valor de salida impide que el auto se mueva.

-kp=1.75 con valor deseado 80 rpm

entonces la salida obtenida es 140 pwm . Este valor de salida impide que el auto se mueva.

-kp=1.875 con valor deseado 80 rpm

entonces la salida obtenida es 150 pwm . Este valor de salida impide que el auto se mueva.

-kp=2 con valor deseado 80 rpm

entonces la salida obtenida es 160 pwm . Este valor de salida hace que el auto oscile entre (0 y 160)pwm.

A la rueda A se le asignó este valor:

-kp=2.25 con valor deseado 80 rpm

entonces la salida obtenida es 180 pwm

A la rueda B se le asignó este valor:

-kp=2.5 con valor deseado 80 rpm

entonces la salida obtenida es 200 pwm .

Se observó que el auto no oscila bien.

---------------------------------------------------------------------------

A la rueda A se le asignó este valor:

-kp=2.375 con valor deseado 80 rpm

entonces la salida obtenida es 190 pwm

A la rueda B se le asignó este valor:

-kp=2.5 con valor deseado 80 rpm

entonces la salida obtenida es 200 pwm .

Se observó que el auto no oscila bien.

---------------------------------------------------------------------------------

A la rueda A se le asignó este valor:

-kp=2.35 con valor deseado 80 rpm

entonces la salida obtenida es 188 pwm

A la rueda B se le asignó este valor:

-kp=2.5 con valor deseado 80 rpm

entonces la salida obtenida es 200 pwm .

Se observó que el auto no oscila bien.

-------------------------

A la rueda A se le asignó este valor:

-kp=1,825 con valor deseado 80 rpm

entonces la salida obtenida es 146 pwm

A la rueda B se le asignó este valor:

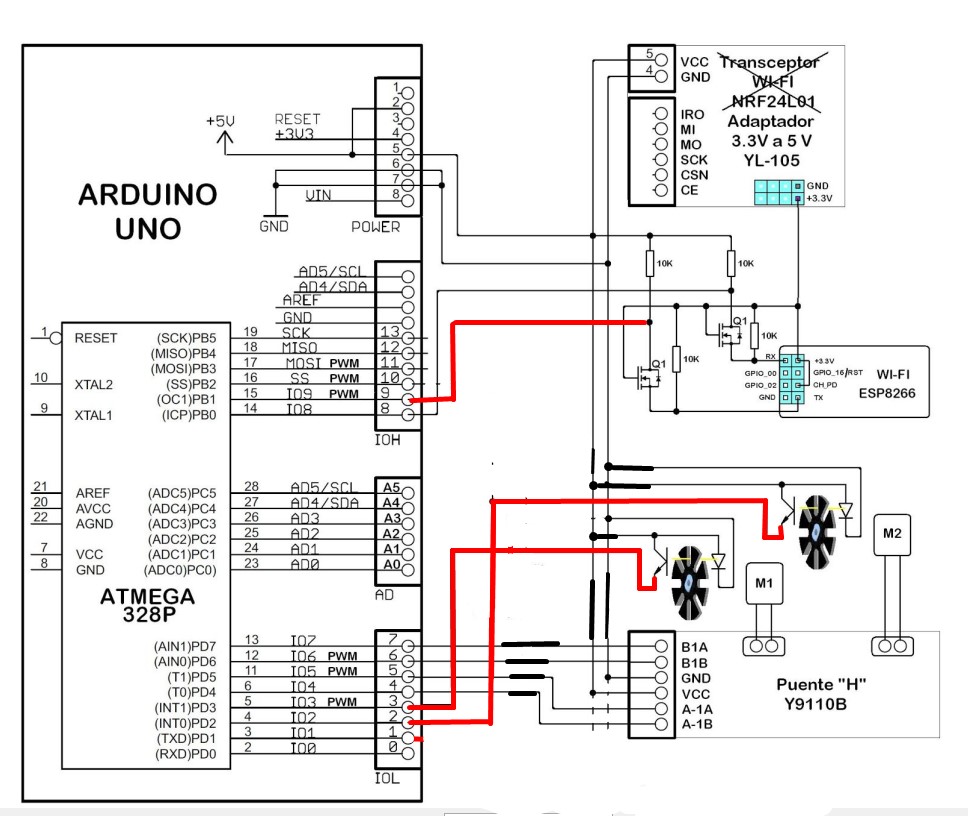
-kp=1,6625 con valor deseado 80 rpm

entonces la salida obtenida es 133 pwm .

Se observó que el auto no oscila bien.

29/10/18

Esquemático final



29/10/18

El siguiente código representa el movimiento del auto, usando el módulo wifi, la bateria y comandos de teclado:

#include <SoftwareSerial.h>

#include <L9110.h>

//-------------------- Variables

String tipo = "Hola \n\r";

int connectionId;

int Intnumero;

int Intaccion;

char inicio;

//--------------------Creando Instancias de Objetos

SoftwareSerial ESP8266(9,8); /\* RX:9, TX:8 OJO revisar conexiones y confirmar si esta bien\*/

L9110 motor(4,5,6,7); /\*Revisar el orden de los pines por que las funciones responden bien si los pines estan bien configurados\*/

void setup()

{// velocidad a elegir 19200 115200

Serial.begin(115200); // la velocidad a elegir para el puerto es 115200

ESP8266.begin(115200); // la velocidad a elegir para el modulo tambien debe ser de 115200

Serial.println("Iniciando...");

Serial.println("\nPrueba de Funcionamiento ESP8266:");

comandoESP("AT+RST"); // resetear el modulo

delay(3000);

comandoESP("AT+CWMODE=2"); // configuro como access point

comandoESP("AT+CIPMUX=1"); // configuro para multiples conecciones

comandoESP("AT+CIPSERVER=1,80"); // encender el servido en el puerto 80

comandoESP("AT+CIFSR"); // nos da la direccion ip

delay(6000);

}

void loop()

{

//Serial.println('.');

if(ESP8266.available()) // comprueba si el esp está enviando un mensaje

{

Serial.println("CONECTADO ");

if(ESP8266.findUntil("+IPD,",".")) // si el modulo encuentra al final "+IPD,","."

{

delay(100); // espere a que se llene el búfer serial (lea todos los datos seriales)

// obtener el ID de conexión para que podamos desconectar

connectionId = ESP8266.read()-48; // restar 48 porque la función read () devuelve

// el valor decimal ASCII y 0 (el primer número decimal) comienza en 48

Serial.println("leyendo pagina...");

// en su tipo de navegador debe escribir http://192.168.4.1/cmd=izqx

if(ESP8266.find("cmd=")) // avance el cursor hasta "cmd="

{

tipo = ESP8266.readStringUntil('x');// retorna la cadena completa leída del buffer serie (por ejemplo cmd=derx), hasta que se detecte el carácter x.

//es decir, que al colocar la x al final del comando que querramos realizar ahi sabe que tiene que captar dicho comando (der,izq,on).

Serial.println(tipo);

// comienzo del arranque de las ruedas, adelante, derecha, izquierda, atras, parar.

if(tipo == "on") // si es adelante

{

Serial.println("adelante");

motor.adelante(240,240,1000);

}

if(tipo == "der") // si es derecha

{

Serial.println("derecha");

motor.derecha(240,240,1000);

}

if(tipo == "izq") // si es izquierda

{

Serial.println("izquierda");

motor.izquierda(240,240,1000);

}

if(tipo == "atr") // si es atras

{

Serial.println("atras");

motor.atras(240,240,1000);

}

if(tipo == "off") // si es parar

{

Serial.println("parar");

motor.frenar(2000);

}

}

//int pinNumber = (ESP8266.read()-48)\*10; // obtengo el primer número, es decir, si es el pin 13 entonces el primer número es 1, luego multiplique para obtener 10

//pinNumber += (ESP8266.read()-48); //obtengo el segundo número, es decir, si el número pin es 13, entonces el segundo número es 3, entonces lo agrego como el primer número

//digitalWrite(pinNumber, !digitalRead(pinNumber)); // toggle pin

}

String closeCommand = "AT+CIPCLOSE="; //cerrar coneccion

closeCommand+=connectionId; // añadir conexión ID

comandoESP(closeCommand); //dejo coneccion

}

}

void comandoESP(String cmd)

{

ESP8266.println(cmd);

if(ESP8266.available()) // comprueba si el esp está enviando un mensaje

delay(1000);

Serial.println(ESP8266.readStringUntil(14)); // si el modulo encuentra al final 14

delay(1000);

}

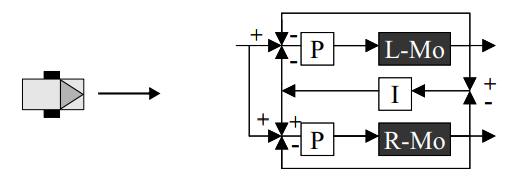
**6/11/2918**

**Propuesta de implementación**

Una mejora de esta estructura de control se muestra en la siguiente imagen como un segundo intento.

Ahora también calculamos la diferencia en los movimientos del motor (posición, no velocidad) y envíamos esto de vuelta a ambos controladores PI a través de un controlador I adicional.

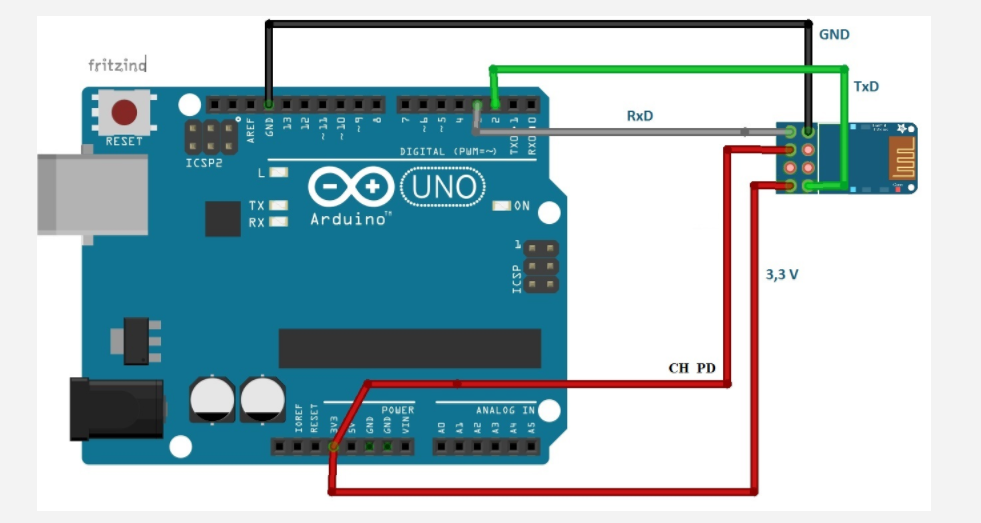
El controlador I integra las diferencias de posición, que posteriormente serán eliminados por los dos controladores PI. Nota los diferentes signos para la entrada de este valor adicional, que coincida con el signo inverso de la correspondiente entrada del controlador I. Además, este principio se basa en el hecho de que todo el circuito de control es muy rápido en comparación con la velocidad del motor o vehículo real.



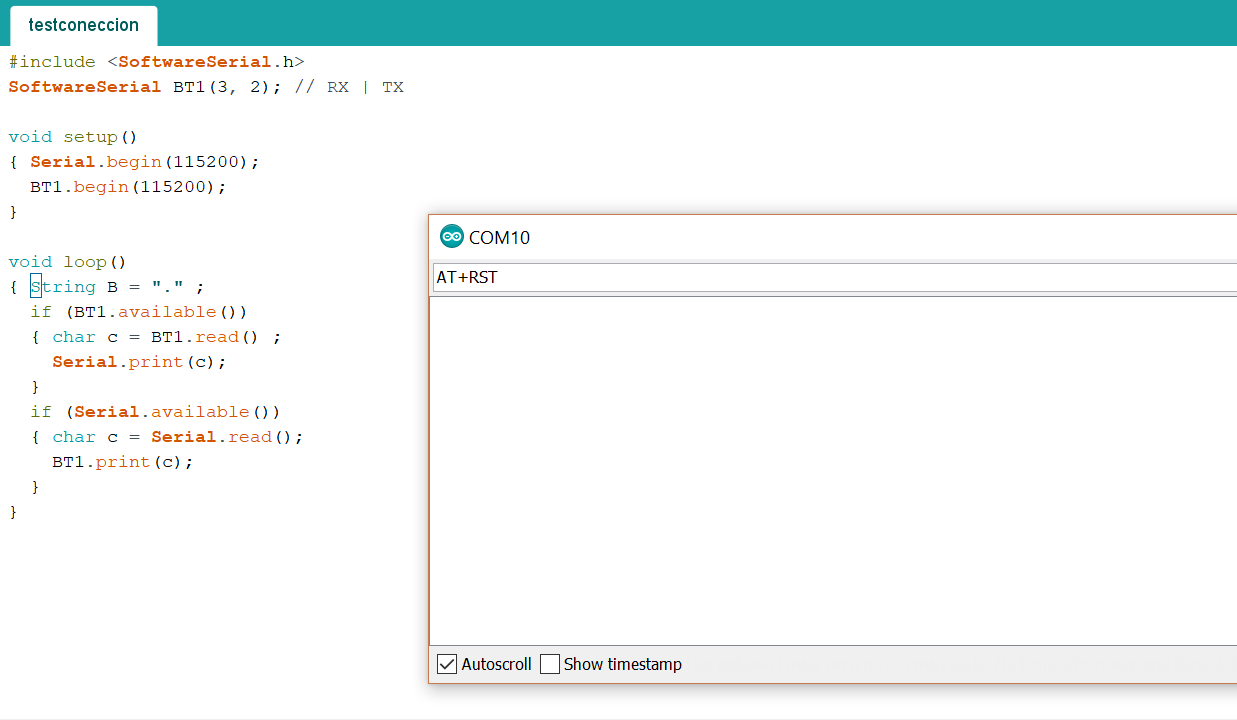
9/11/18

Pruebas de funcionamiento del esp8266

Realizando las siguientes conexiones

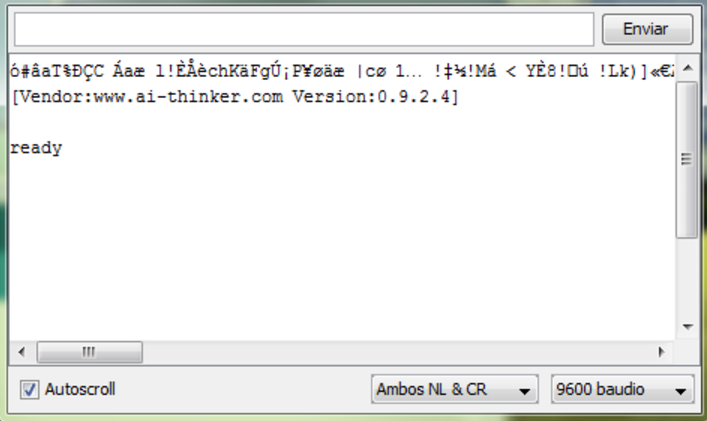


y con el siguiente código para poner a prueba el funcionamiento del módulo wifi

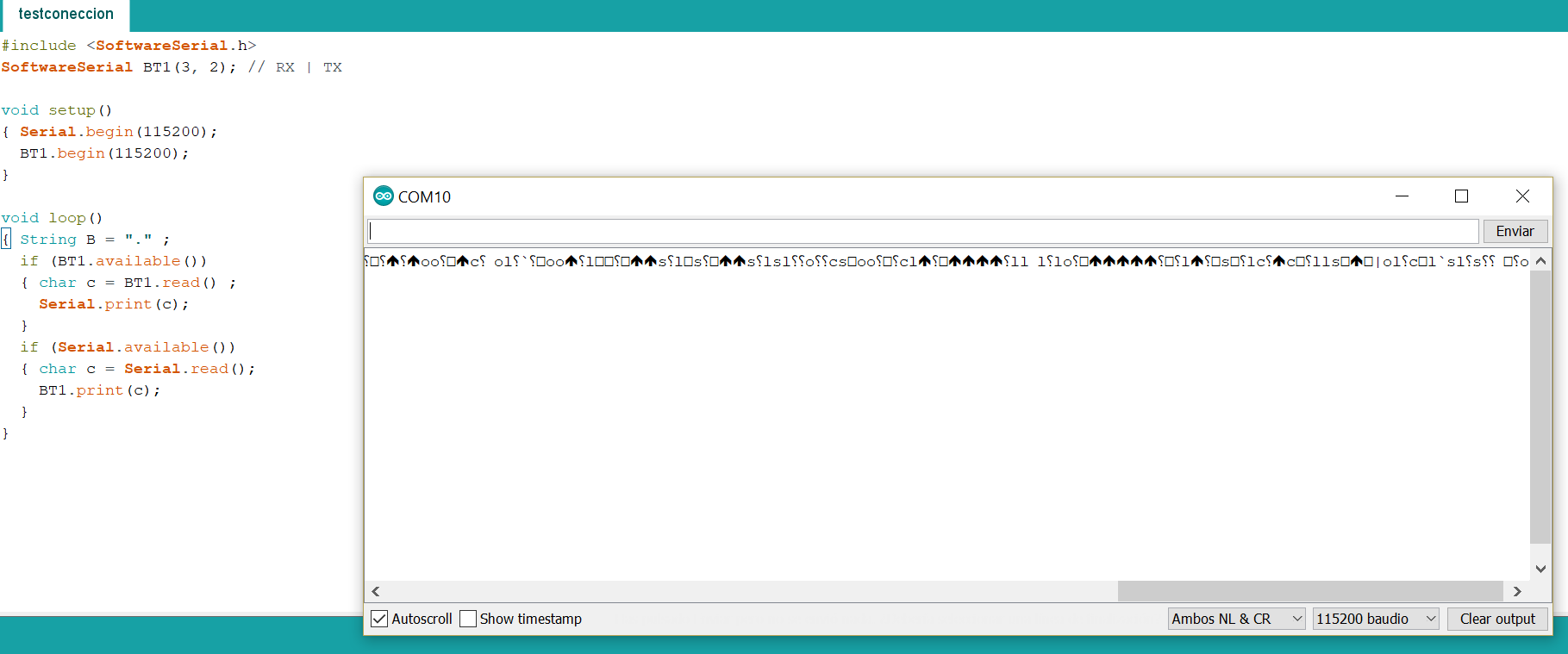


No se obtuvieron respuestas, del módulo al comando ingresado AT+RST se repitieron la misma secuencia de prueba con una tasa de 9600 y de 115200 y en ambos casos la respuesta es la misma que la imagen anterior.

lo que se esperaba recibir como respuesta



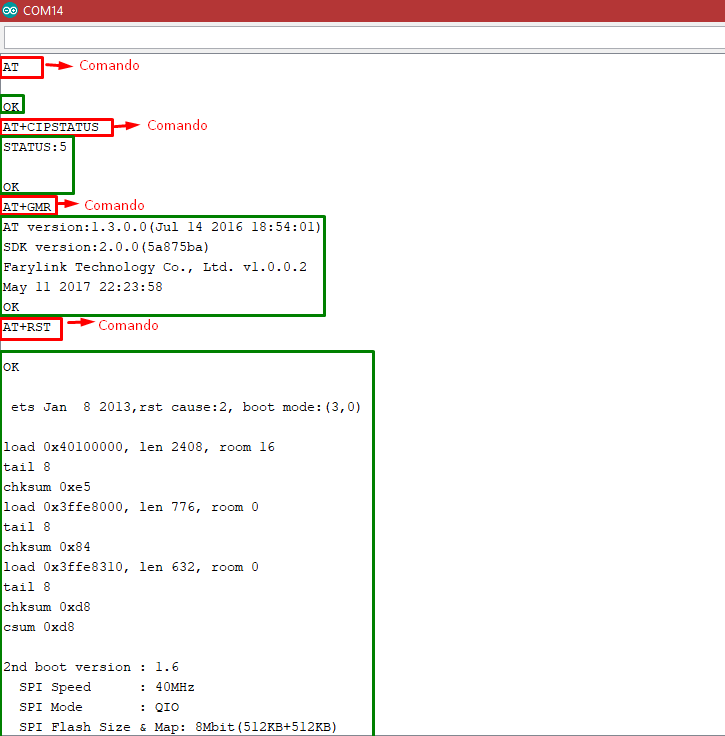
cuando se desconecta la alimentación unos 5 segundos y se la vuelve a conectar a modo de reinicio manual se ve lo siguiente:

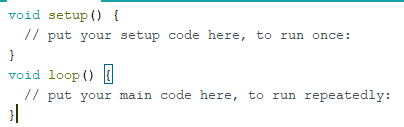


20/11/18

Utilizamos el nuevo ESP826601 no integrado en el auto y verificamos su funcionamiento con comandos AT y con sketch vacío.

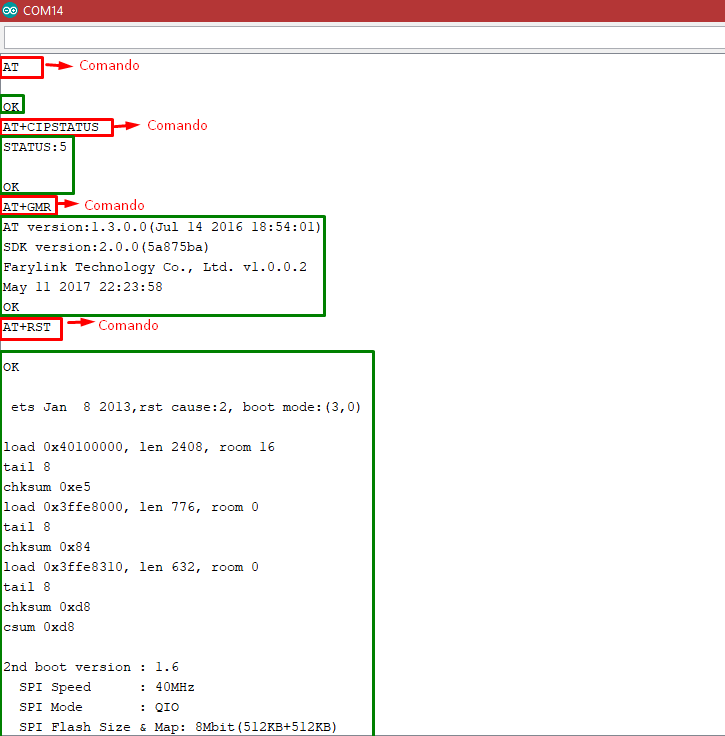
Nota: La coneccion de Tx y RX es directa (Tx del arduino con el TX del ESP).

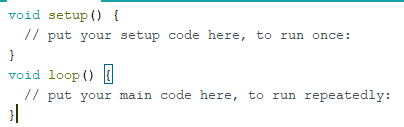




Utilizamos el nuevo ESP826601 integrado en el auto y verificamos su funcionamiento con comandos AT y con sketch vacío.

Nota: La coneccion de Tx y RX es directa (Tx del arduino con el TX del ESP).





# 30/10/2018

**Diseño de la página web para control del autito:**

para la presentación de página se tomó en cuenta el siguiente diseño

se investigó el uso de json y js y se consiguió documentación para el uso de una determinada función para obtener un object json mediante :

## $.get(URL, funcion)

En este segundo caso, estamos pasando dos parámetros, el primero la URL de la solicitud Ajax a realizar y el segundo una función con el código a ejecutar cuando se reciba la respuesta, que incluirá todas las acciones a realizar cuando se reciba. En esa función a su vez recibimos varios parámetros, siendo el más importante el primero, en el que tendremos una referencia al resultado de la solicitud realizada. Lo vemos con un ejemplo:

$.get("contenido-ajax.html", function(respuestaSolicitud){

alert(respuestaSolicitud);

})

En este caso hacemos una solicitud al archivo "contenido-ajax.html". Luego, cuando se reciba la respuesta se ejecutará el código de la función. En la función recibimos un parámetro " respuestaSolicitud", que contendrá el código HTML devuelto por el servidor al solicitar esa página por Ajax. Como se puede ver, en la función simplemente mostramos en una caja de alerta el contenido de la respuestaSolicitud.

Ese código en marcha se puede [ver en una página aparte](http://www.desarrolloweb.com/articulos/ejemplos/jquery/ajax/ajax-get.html).