Control remoto de vehículo, joystick, Arduino Wemos y comunicación ESP-NOW

Introducción:

Resumen del trabajo:
.- Varios entradas analógicas a través de un solo puerto.
.- Wemos, especificaciones eléctricas.
.- Protocolo de comunicaciones ESP-NOW.
.- Circuito L298N. Especificaciones y pinout del mismo.
.- Montaje vehículo con dos motores DC

Todo parte de la idea de poder mover una silla de ruedas para personal discapacitado vía remota y poder acompañarlos sin necesidad de empujar la misma. Como ejemplo de funcionamiento, he creado este proyecto. Posteriormente se pueden cambiar los circuitos de salida y los motores, por otros de mayor potencia y acoplar a las ruedas de la silla un sistema mecánico que la mueva.

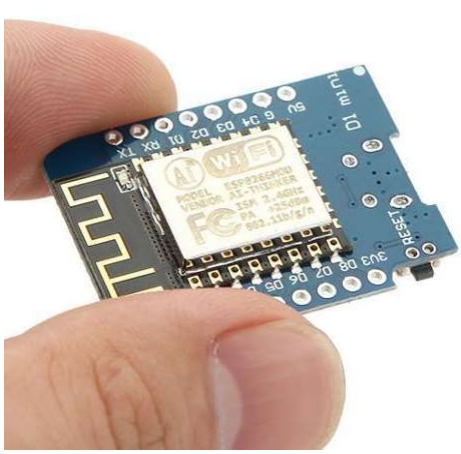
En este trabajo explico como tomar varios valores analógicos e introducirlos en un único puerto A0 de una placa Wemos. Los valores provenientes de un joystick, se transmiten de forma rápida, segura y fácil por medio de Wifi usando el protocolo ESP-NOW. En el vehículo, otra Wemos recibe los datos y acciona dos motores DC para controlar la dirección del vehiculo.

Quizás alguien se pueda plantear que las cosas expuestas de estos trabajos, se puedan conseguir de forma fácil y barata en alguna web, pero el hecho de hacerlo tu mismo y con componentes de bajo precio siempre es una satisfacción cuando lo ves funcionar. Aparte de eso, me conformo con que a una persona le guste o le aclare algún concepto o duda.

Intentaré explicar los conceptos usados para mejor comprensión del trabajo. Quizás a algunos le parezca interesante alguna parte del mismo.

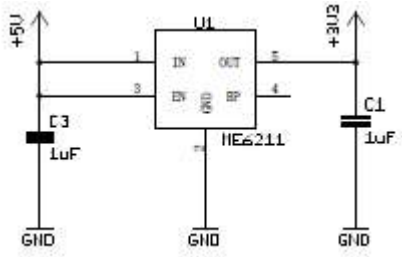
Placa de desarrollo Arduino Wemos:

Estamos hablando de una pequeña placa de desarrollo con amplias posibilidades:

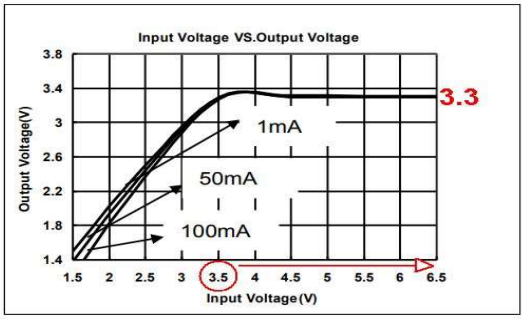


Con ella podemos realizar proyector IoT, análisis de datos y envío a través de las redes y otras muchas cosas, aprovechando la capacidad Wifi de las mismas. En otro proyecto que he realizado, creo una red wifi propia y puedo abrir una cerradura remota, mediante una clave tecleada desde nuestro smartphone, que también he publicado. La diferencia respecto al anterior es que en vez de usar protocolo HTLM para la comunicación, uso la característica muy poco publicada de la comunicación WiFi del tipo ESP-NOW entre dos dispositivos, por ser fácil, rápida, segura (encriptada) y sin necesidad de emparejamientos a la hora de actuar (solo al configurar el sketch de Arduino). Mas adelante, a la hora de explicar el sketch, comentaré los detalles a tener en cuenta.

La placa dispone de una **entrada** de alimentación de 5v en el pin correspondiente (o por USB) y de una entrada de GND. Dicha alimentación no tiene porque ser 5v , ya que lleva un regulador de voltaje que lo convierte en 3.3v, que es realmente el voltaje de trabajo. En la datasheet de la Wemos podemos verlo y adjunto también una imagen de la datasheet del regulador:



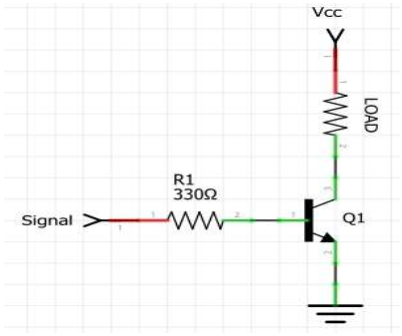
（2）Input VoltageVS.Output Voltage（**Ta = 25 °C**）
ME6211C33M5G

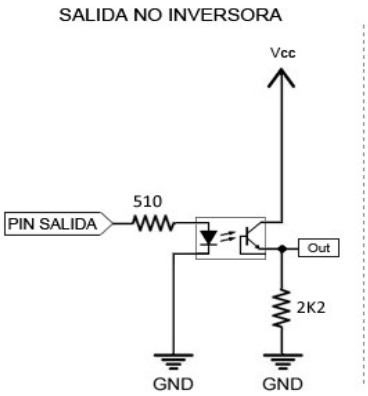


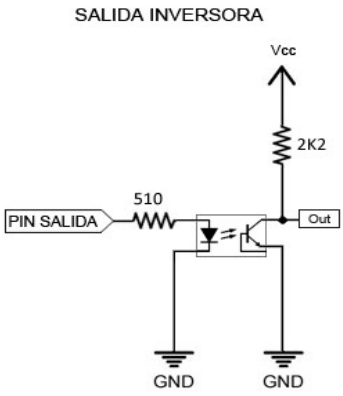
Según el link de las especificaciones del ESP8266, podría trabajar incluso a 3v, pero conviene alimentarlo con un voltaje superior a 3.5v, para que a la salida del regulador interno tengamos un mínimo de 3v. En dicho link se puede ver otros detalles técnicos que amplian esta información.
https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

La Placa también dispone de 9 entradas/salidas digitales (D0-D8). Todas tienen la capacidad de poder trabajar con salidas del tipo PWM, bus I2C, etc.

Detalle a tener muy en cuenta a la hora de conectar algo a la salida de los pines digitales, para iluminar leds, activar relés, etc. La corriente máxima que puede entregar un pin Digital **es de 12mA**. Si se necesita entregar mas corriente, debemos intercalar entre el pin y el dispositivo un transistor o un opto acoplador de mayor potencia. Por ejemplo:







Con una resistencia en serie con la salida de 330 ohms, se entrega una corriente de 10mA, por lo que si es posible, aumentar el valor de las resistencias. Hay en muchas webs la recomendación de una resistencia de 330 ohm en serie con los leds Yo recomiendo usar resistencias mas altas. Si ilumina el led a nuestro gusto, no necesitamos sumar mA al trabajo Cualquier ahoro de energía siempre es bueno.

NOTA: en los pines digitales, podemos dar valores PWM entre 0 y 1023. En Arduino Uno, entre 0 y 254.

La placa Wemos también dispone de una entrada digital A0, para análisis de datos analógicos. Hay que tener en cuenta dos cosas. La primera es que NO se le puede aplicar un voltaje superior a 3.3v directamente, ya que se deterioraría. Si se quiere medir un voltaje superior, hay que intercalar un divisor de voltaje externo. Los valores de dicha entrada son de 0 a 1024.

Otras características:

-Salida de 3.3v para alimentar circuitos exteriores. Máxima corriente 12mA por pin.

-Conector micro USB para la carga del firmware y alimentación de 5v

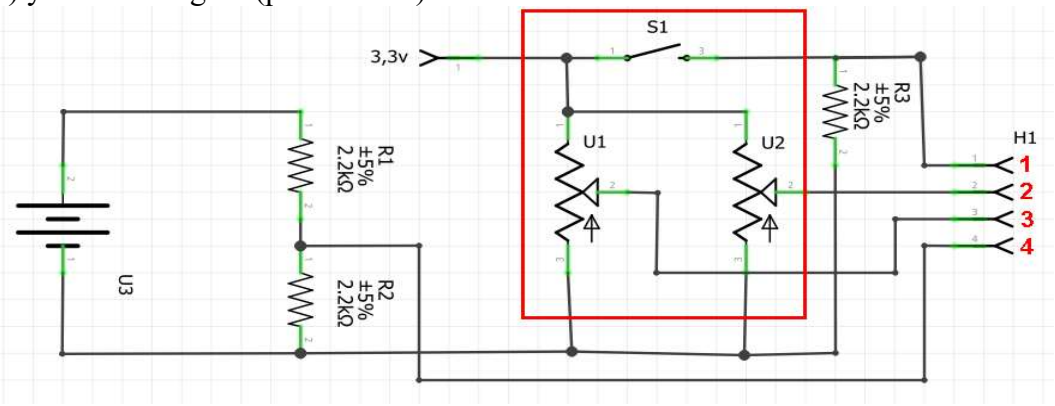
-Pulsador de Reset.

Hay muchos tutoriales de como configurar el IDE de Arduino para trabajar con este tipo de placa, así como las librerías necesarias. No voy a entrar en ello para no alargar demasiado este trabajo.

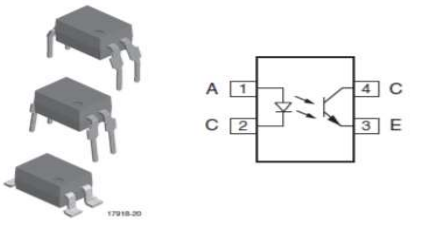
Circuito del Joystick (mando a distancia):

Me gusta la placa de desarrollo Wemos, ya que tiene poco tamaño, es barata y tiene muchas posibilidades. Como solo dispone de una entrada analógica A0, surge el problema de querer captar varios valores analógicos al mismo tiempo. Para mi caso en concreto, un joysick está formado por dos potenciómetros con salidas individuales analógicas y un pulsador. Además, quiero analizar el valor actual de la batería que uso en el mando a distancia, por lo que ya necesitamos tomar 3 valores analógicos distintos.

En el siguiente esquema, creado con Fritzing, tenemos a la izquierda un divisor de voltaje. Si la batería es de mas de 3.3v, la entrada analógica corre riesgo de averiarse, por ello conviene reducir el voltaje para su análisis. Voy a usar una batería de 3.7v, por lo que cuando está cargada completamente es de aproximadamente 4v y debido al divisor de voltaje, en el pin 4 de H1 tenemos 2v (variable dependiendo del estado de la batería). A la derecha tenemos un joystick básico, formado por dos potenciómetros y un pulsador (R3 es externa al joystick). Se alimentan con los 3.3v que proporciona la Wemos. En este esquema general, tenemos 3 valores analógicos (pines 2, 3 y 4 de H1) y un valor digital (pin 1 de H1).

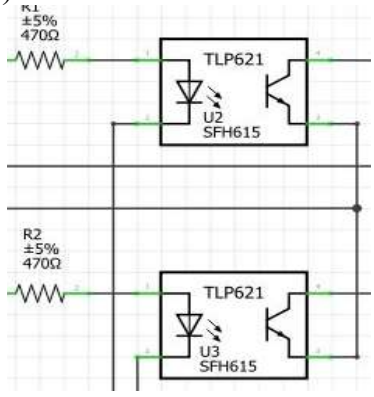


Para poder analizar en la placa Wemos los 3 valores analógicos, recurrimos a unos pequeños opto-acopladores, el chip SFH615A o TLP621. Es muy básico su funcionamiento para este trabajo. En el pin 4 del chip pongo uno de los valores analógicos a analizar. Todos los pin 2 a GND. Todos los pin 3 unidos y a A0 y cada uno de los pin 1 a una salida digital a través de un resistor, las cuales voy activando sucesivamente y dependiendo cual active y leyendo el valor en A0, asigno a cada valor una variable (pot 1y pot 2 del joystick y batería).



DESCRIPTION

The SFH615A feature a variety of transfer ratios, low coupling capacitance and high isolation voltage. These couplers have a GaAs infrared diode emitter, which is optically coupled to a silicon planar phototransistor detector, and is incorporated in a plastic DIP-4 package.

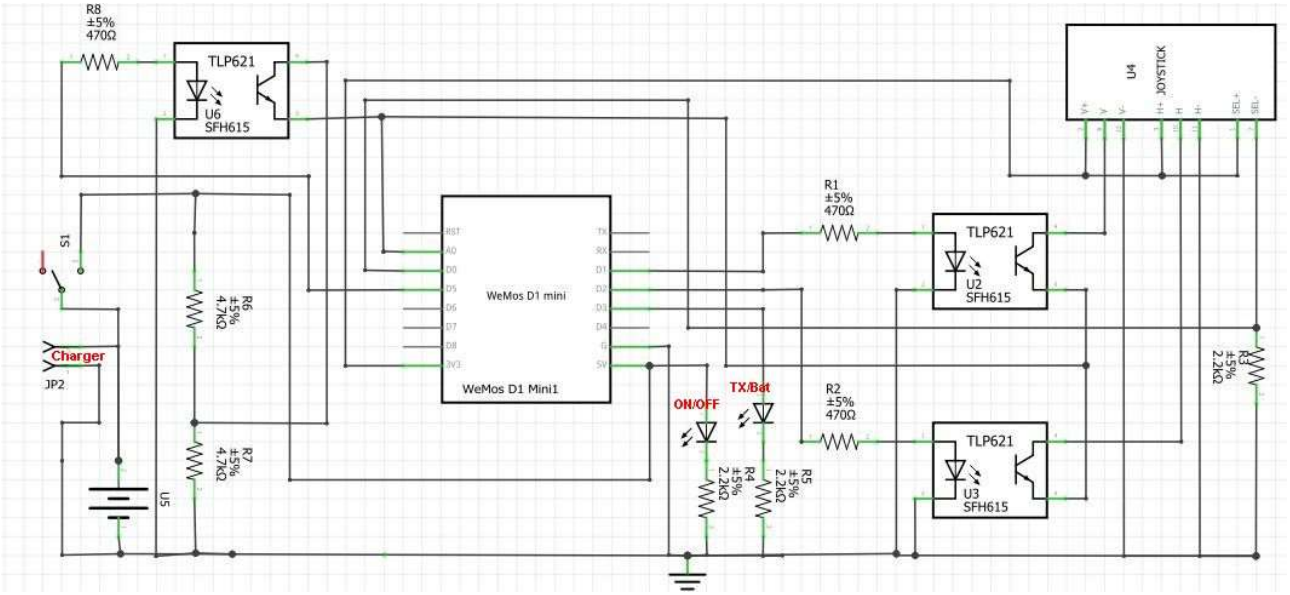


Hay que tener en cuenta que no podemos conectar la salida digital de la Wemos directamente al pin 1 del TLP621, ya que se deterioraría dicha salida digital. Cada pin digital en Wemos puede suministrar unos 12mA. Por ello, intercalamos una resistencia suficiente para activar el led interno. Con 470 Ω , es suficiente para activarlo y solo supone 7 mA.

Al querer introducir 3 valores analógicos mediante este sistema, usamos 3 salidas digitales para poder activarlas. Si queremos introducir mas valores analógicos por A0, podemos usar otras salidas digitales más o podemos seguir usando solo 3 salidas digitales, añadiendo al circuito un demultiplexor y dando valores binarios a las entradas, conseguimos hasta 8 posibles valores digitales.

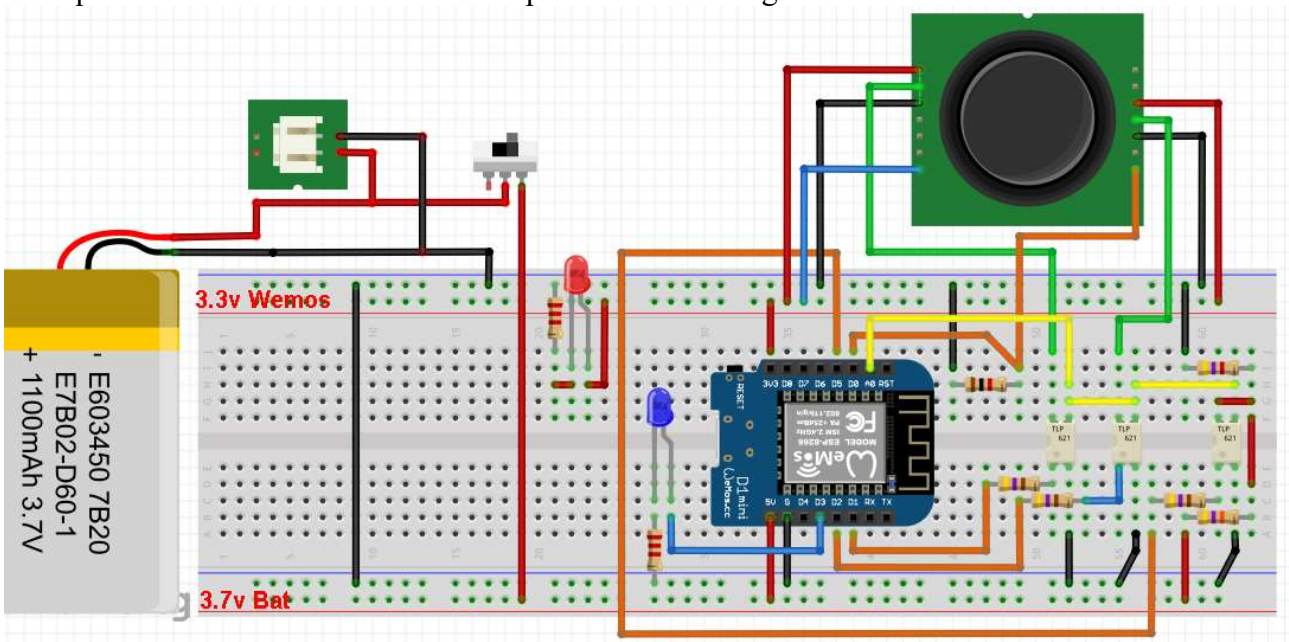
Añadimos al mando a distancia 2 leds, uno para reflejar “Power ON” y el otro para el estado de la batería y “Transmisión OK”.

Añado al circuito un interruptor para la batería y un conector para poder recargar la misma sin tener que quitarla (aviso: APAGAR PARA RECARGAR para evitar dañar el regulador ME6211 de la placa Wemos). Con todo lo anteriormente explicado, el circuito completo del mando a distancia con joystick es el siguiente:



Explicación para el posterior desarrollo en el IDE de Arduino:
En A0 recojo los valores de los potenciómetros y del nivel de la batería.
En D0 pasa a HIGH cuando se pulsa el botón del joystick (“parada de emergencia”)
Si activo D1, leo el estado del potenciómetro vertical del joystick en A0.
Si activo D2, leo el estado del potenciómetro horizontal del joystick en A0.
Si activo D5, leo el estado de la batería en A0. NOTA: en un principio lo puse en D4, pero me daba problemas al flashear el programa desde el IDE de Arduino, por lo que la pasé a D5
La salida D3 se usará para el led de Actividad (azul). Dicho led se enciende cuando hay movimiento de joystick y la transmisión ha sido correcta. Cuando está en reposo nos indica el estado de la batería (1 parpadeo entre 3.6 y 3.5v, 2 parpadeos entre 3.5 y 3.4v y 3 parpadeos por debajo de 3.4v).
El led rojo indica Encendido/Power ON.
S1 es el interruptor de encendido. Conviene tenerlo apagado cuando se realiza la carga de la batería o si hago modificaciones en el software (5v a través del USB).

El esquema del circuito montado en una protoboard es el siguiente:

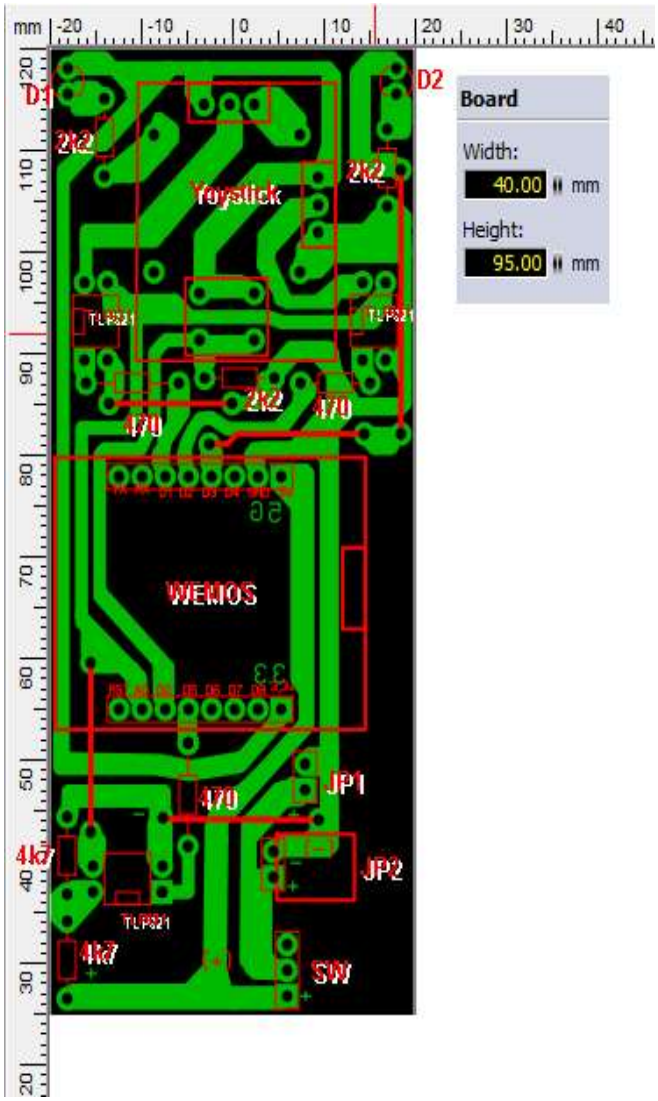


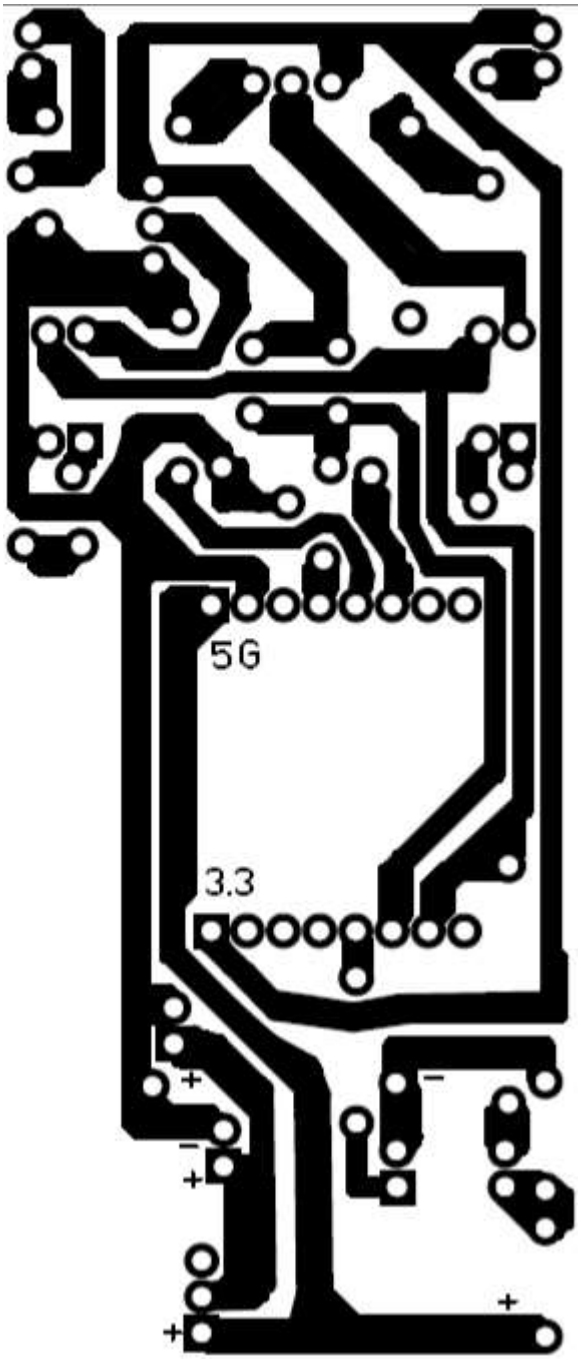
La línea inferior positiva es el voltaje de la batería. La línea superior positiva es la salida de 3.3v de la Wemos

He diseñado la siguiente placa de circuitos con Sprint-Layout 6.0 para la conexión del joystick, opto acopladores, Wemos y otros. Indico las medidas por si alguien la quiere realizar (40x95mm).

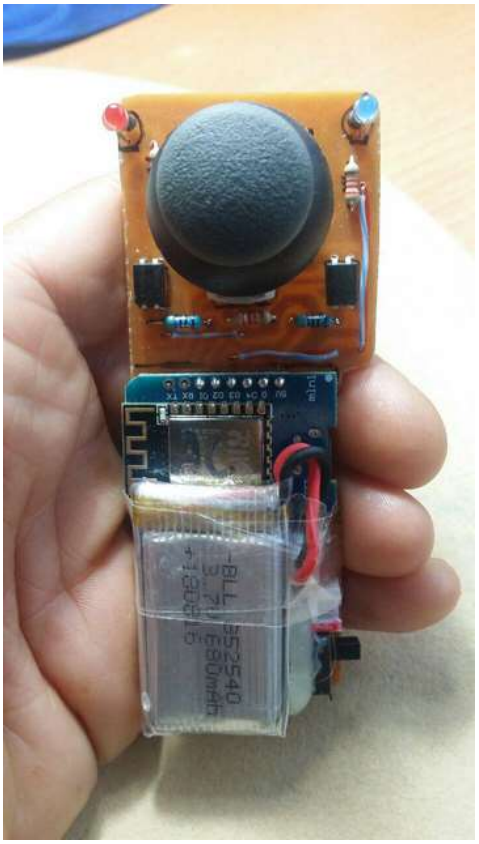
Hay que tener cuidado con el pin 1 de los TLP621. Van soldados al terminal cuadrado y en la posición indicada visto desde la cara de los componentes.

La parte de la placa próxima a los conectores y Wemos, la recorto posteriormente, así queda de forma cómoda el agarre del mando, el encendido y las conexiones externas.





Las fotos del mando a distancia. En los bordes, las conexiones USB, el conector de carga de la batería y el interruptor de ON/OFF.
Fácil de sujetar, aunque sea un poco grande. Me falta realizar una caja a medida para el mismo con la impresora 3D:

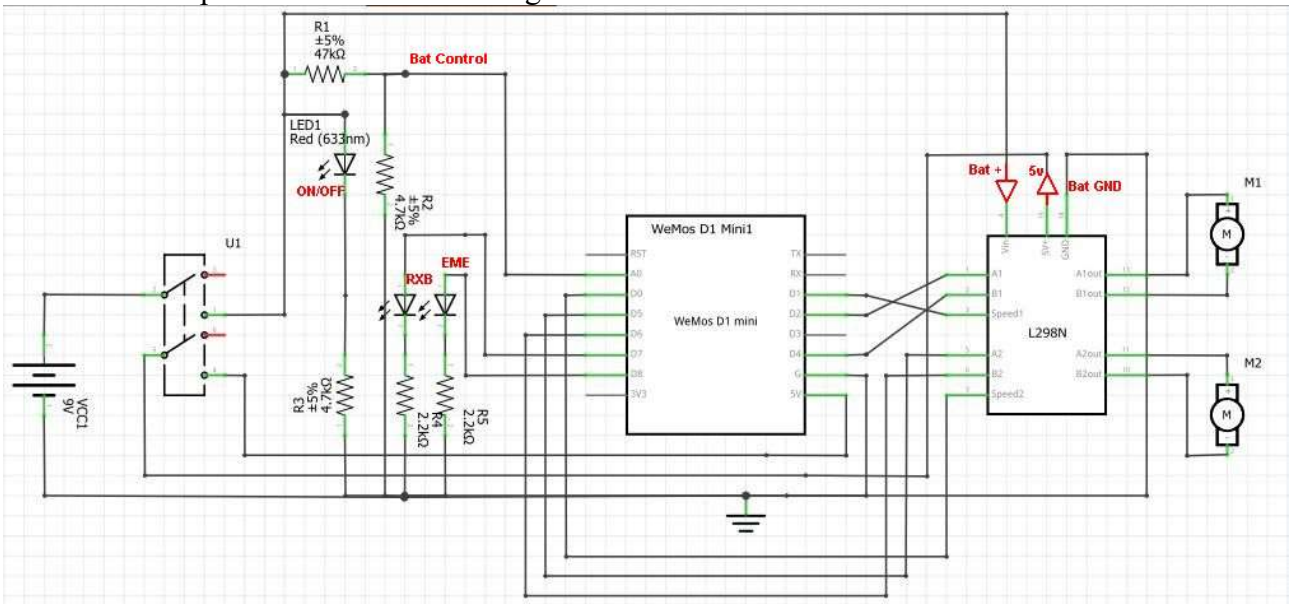


Circuito del receptor (Motores):

Está compuesto por otra placa Wemos, donde recibo la data del joystick o control remoto y activa las señales necesarias hacia un L298N (doble puente en H) y controlar dos motores, hacia adelante y hacia atrás, con control de dirección. Como complemento del circuito, 3 leds, uno para power ON, otro para la transmisión de datos y un tercero como indicativo de “parada de emergencia”. Aprovecho estos dos últimos (parpadeando) para la indicación del estado de la batería del vehículo.

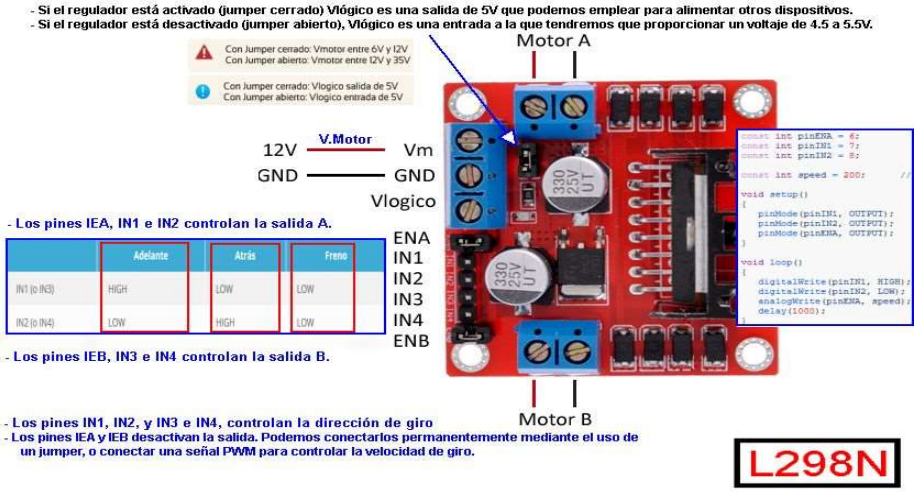
Control de estado de la batería: Lo primero a tener en cuenta es que la batería que estoy usando es de 9v. Intentar medir la misma en A0 directamente, supone deteriorar el puerto, ya que el máximo valor que se le puede aplicar es de 3.3v. Para evitarlo, ponemos también otro divisor de voltaje, esta vez mas descompensado que en el mando a distancia y reducir el valor en A0. Para este caso, utilizo un resistor de 47k en serie con otro de 4k7. En el punto central es donde tomo la referencia a medir. "Bateria baja", entre 7v y 5.5v, 1 parpadeo del led de “Emergencia”. "Bateria MUY baja" (por debajo de 5,5v, 3 parpadeos del led “Recepción ok”)

El circuito completo del vehículo es el siguiente:



Debido a que este circuito está montado sobre un vehículo, no he querido complicar mucho el sketch de Arduino. Simplemente recibe los datos del joystick via wifi ESP-NOW y los convierte en señales de control para los motores. Eso facilita a que en futuros cambios de software o modificaciones de trayectoria, se realicen solo en el mando a distancia (joystick) en vez de en ambos.
No he realizado ninguna placa de circuitos especial. Tan solo una provisional para los leds y sus resistencias.

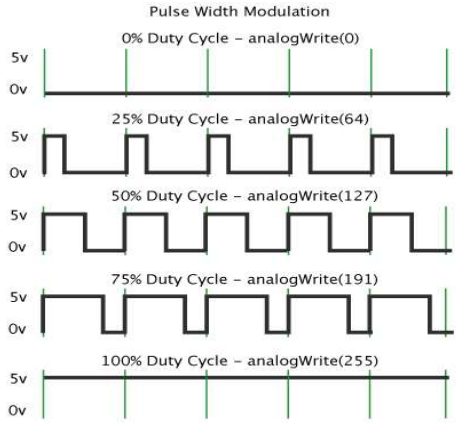
L298N (doble puente en H)



Esta es una pequeña descripción del circuito que controla los motores DC que mueven el vehículo. - Conectores A y B (azules de 2 pines). Son las salidas de corriente hacia los motores. Si tras las pruebas, el motor gira al lado contrario del que deseamos, simplemente invertir los pines del mismo Conector de Power (azul de 3 pines). Es la entrada de corriente al circuito. Como el mismo puede ser alimentado entre 6 y 36 voltios, hay que tener muy en cuenta el jumper o puente que hay junto al conector. Si lo alimentamos con un voltaje entre 6 y 12v, el puente se deja PUESTO y en Vlogico tenemos una **salida** de 5v hacia la Wemos (como en este trabajo). Si el circuito se alimenta con un voltaje superior a 12v, hay que quitar el puente para que no se dañe el convertidor DC-DC que lleva y si queremos que funcione su circuitería lógica, deberemos llevar un cable de 5v externo hacia el circuito (5v input). En mi caso, como utilizo una batería de 9v, lo dejo puesto y me sirve para alimentar la placa Wemos a través del pin 5v. GND viene del negativo de la batería y va también a G de la Wemos y a los leds.

Conector de Control (6 pines). Tiene dos partes. ENA, IN1, IN2 controlan el motor conectado en A y ENB, IN3, IN4 que controlan el motor conectado en B. En la tabla de la figura anterior se indica los niveles de las señales que debe tener para poner en movimiento los motores, adelante, atrás o frenado. En ENA y en ENB hay unos puentes. Si los dejamos puestos, el L298N pondrá los motores al voltaje de entrada Vm en el sentido indicado, sin ningún control de velocidad ni de regulación de voltaje. Si los quitamos, usaremos dichos pines para recibir una señal PWM desde la placa Wemos y así controlar la velocidad de cada motor. En Arduino se consigue mediante un comando analogWrite(). En la placa Wemos, todas los puerto D tienen esa capacidad.

En la figura del L298N hay un recuadro con un pequeño sketch para **Arduino UNO**, que hará girar el motor A hacia adelante a un voltaje cercano al 75% de Vm.



La gráfica anterior a este texto, explica la relación de analogWrite() con la forma de salida en los pines para Arduino UNO. En la Wemos, el 100% se consigue con analogWrite(1023) y al 50% sería analogWrite(512).

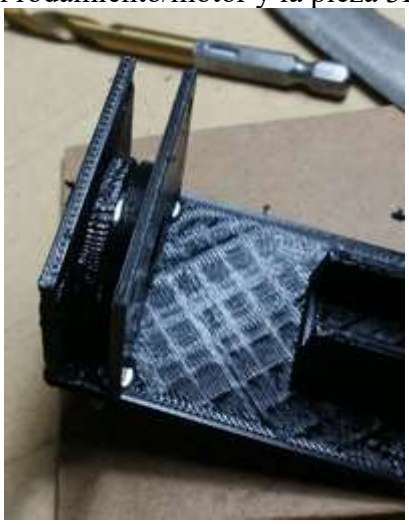
A la hora de realizar este proyecto, hay que tener muy en cuenta los posibles valores PWM de ENA y ENB que se suministran mediante el comando analogWrite, ya que dependen del valor del voltaje de la batería y del voltaje de los motores. En este caso utilizo una batería de 9v (Vm) y motores de 6v. Al ir aumentando la señal PWM en ellos, el voltaje del motor asciende, pero no comienza a moverse hasta que llega a un valor determinado, por lo que en las pruebas, ***se debe establecer ese mínimo PWM que lo haga mover*** a baja velocidad. Por otra parte, si ponemos la señal PWM al máximo, le damos al motor el voltaje Vm de la batería (9v) y se puede dañar el mismo, por lo que en las pruebas, debemos medir el voltaje y ***establecer ese máximo PWM para que no se deteriore*** y como mucho proporcione los 6v máximo. Ambas cosas, como ya comentaba anteriormente, en el sketch de Arduino del mando a distancia.

Montaje del vehículo:

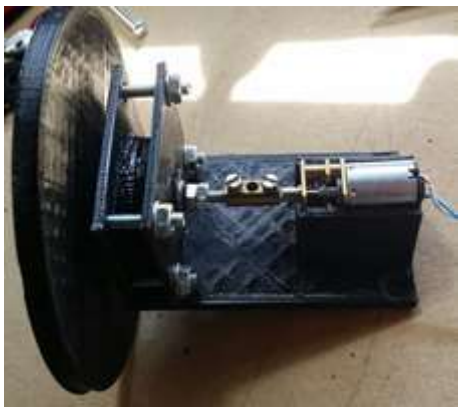
Tengo que reconocer que el montaje es un poco casero, pero efectivo. Quizás diseñe e imprima en 3D un modelo mas bonito, pero este modelo “casero” tiene la ventaja de ver mejor el funcionamiento. Existen una serie de motores, con reductora incluida y ruedas para acoplar, a bajo precio. Yo he usado lo que tengo a mano.

Para el montaje, he impreso en 3D unas piezas, ruedas, soporte de rodamiento/motor y unos casquillos y uso tornillería de 3mm de diámetro para unir las piezas. Para la unión del motor al tornillo eje, he usado los contactos de una regleta de conexión eléctrica cortando el plástico externo. Al montar las ruedas, conviene pegar el tornillo a la rueda, para evitar que patine al girar.

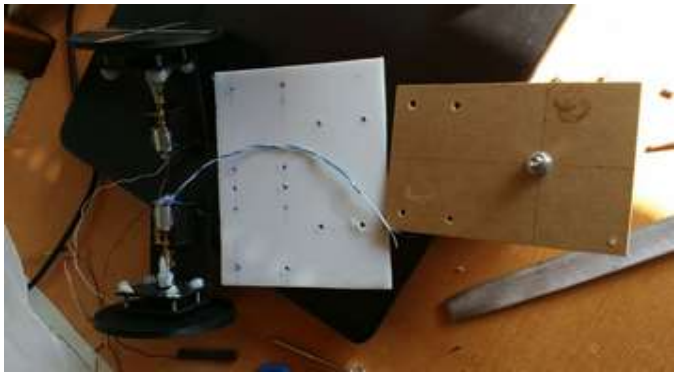
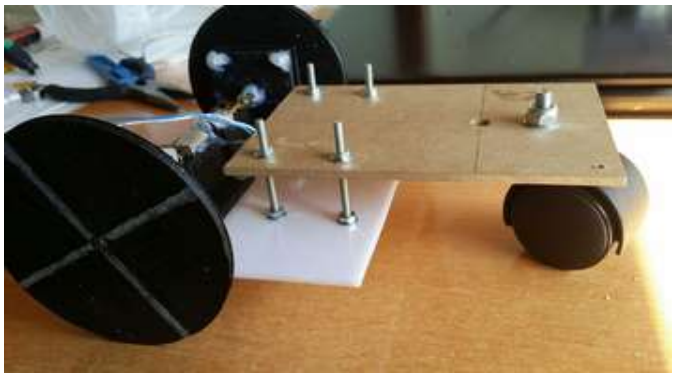
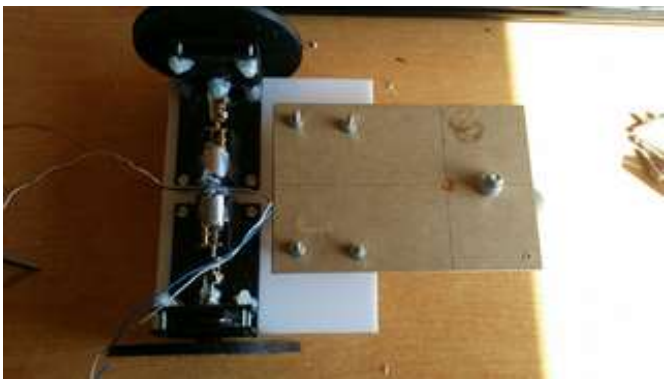
La siguiente muestra el soporte del rodamiento/motor y la pieza 3D que lo sujeta



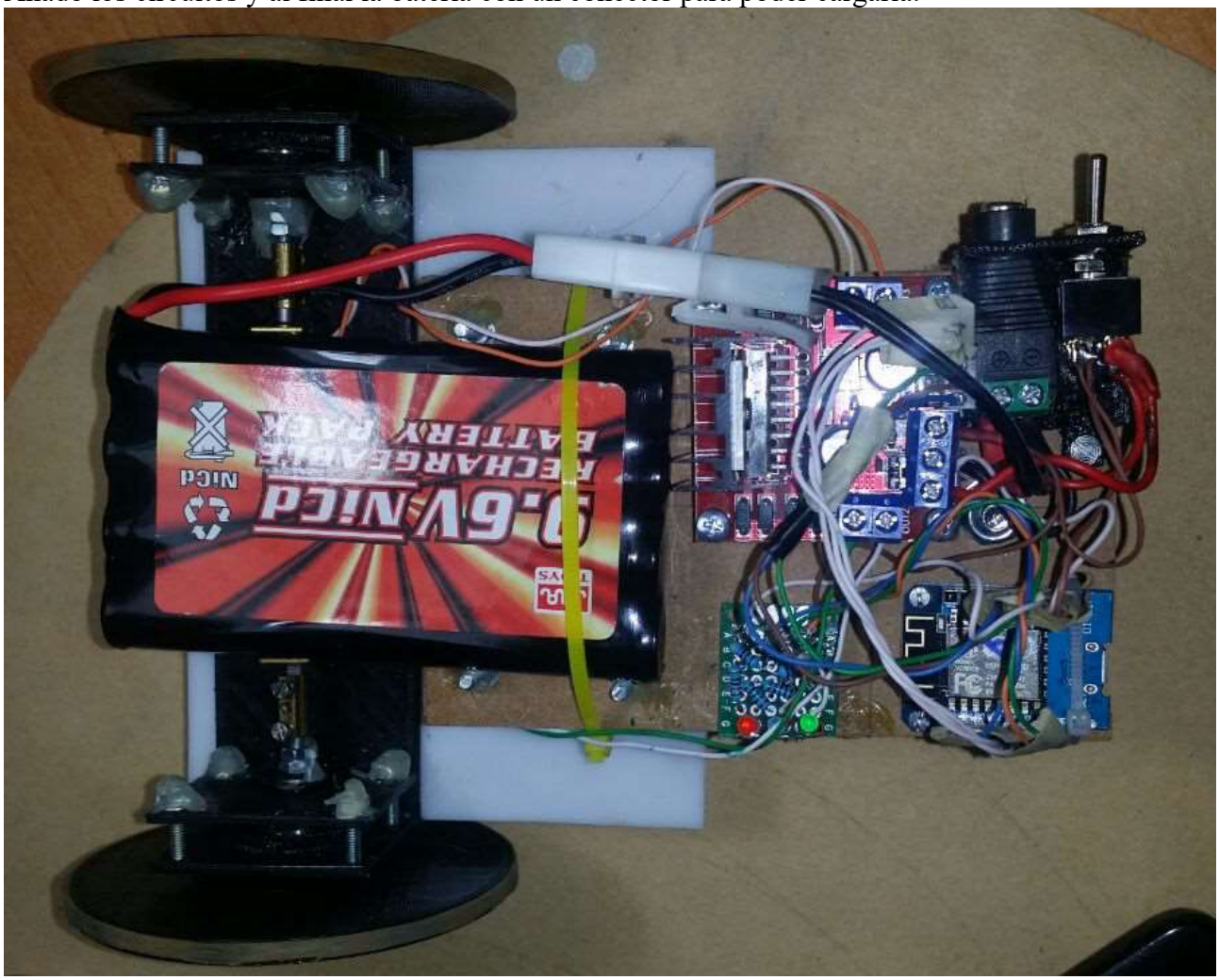
Monto la rueda. Tomo las medidas, corto el tornillo que sobra y los uno:



Una vez realizado el montaje de los dos conjuntos motriz, los sujeto a una plataforma (blanco). Les uno otra plataforma para soporte de los circuitos y la rueda trasera. La diferencia de altura la marca el tipo de rueda que pongamos, para mantener el vehículo horizontal. La distancia entre la rueda trasera y la primera plataforma nos debe asegurar el giro de la misma, por eso tuve que corregir el primer agujero, como veis en las fotos.

Añado los circuitos y al final la batería con un conector para poder cargarla.

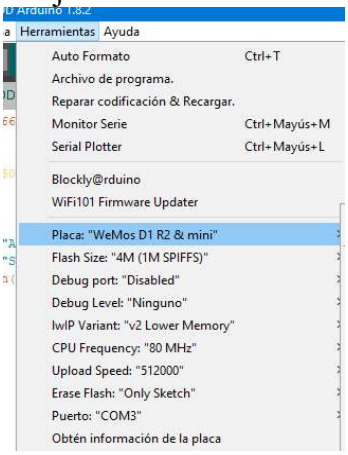


Como veis, no es un gran diseño. Mi intención es aplicar este sistema a una silla de ruedas como comentaba al principio de este trabajo. Pero ya que lo tengo desarrollado, posiblemente diseñe un tipo de vehículo mas elegante.

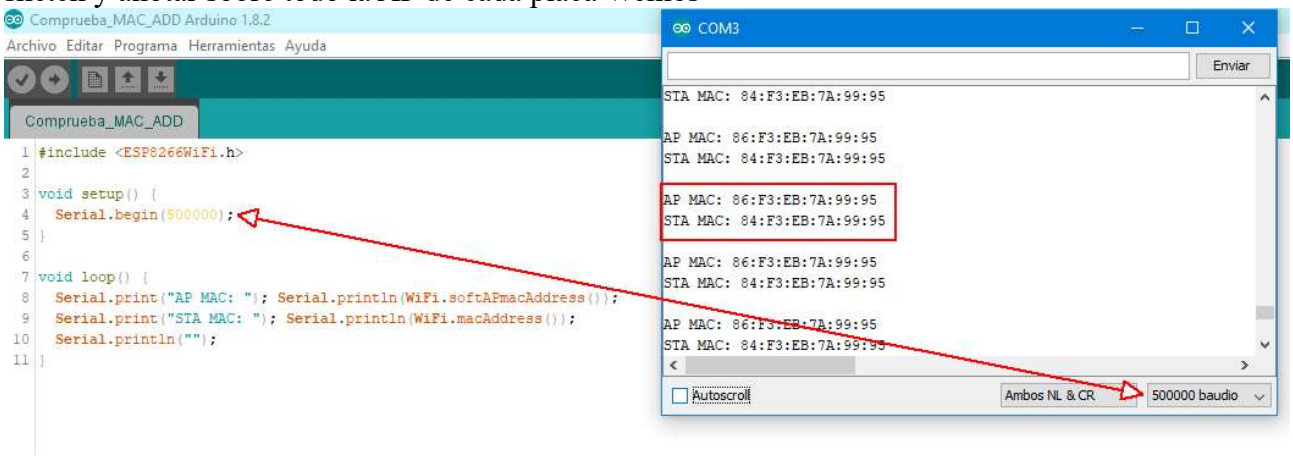
Y ahora pasamos a la explicación del sketch de Arduino que he realizado.

Arduino

Como escribí al principio, no puedo extenderme mucho y prescindo de como configurar el IDE de Arduino, librerías y como debe reconocer la placa Wemos para poder trabajar con ellas. Solo unos datos:
.- En Preferencias, Gestor de URLs adicionales:
http://arduino.esp8266.com/stable/package_esp8266com_index.json
.- En Herramientas (Tools), Gestor de tarjetas:



Como paso previo e imprescindible antes de trabajar con el protocolo ESP-NOW, debemos cargar este pequeño sketch en las Wemos con las que vamos a trabajar, para saber la AP MAC de las ESP8266 que llevan integradas. En Herramientas, Monitor Serie podemos ver el resultado del sketch y anotar sobre todo la AP de cada placa Wemos



Tengo la costumbre de al recibir las que compro, marco las bolsitas y la placa con dicho dato:

Una vez con la AP MAC de las placas, comienzo a hablar del protocolo ESP-NOW desarrollado por Espressif:

"ESP-NOW permite un control directo y de baja potencia de las luces inteligentes, sin la necesidad de un enrutador. Este método es energéticamente eficiente y conveniente.
ESP-Now es otro protocolo desarrollado por Espressif, que permite que múltiples dispositivos se comuniquen entre sí sin usar Wi-Fi. El protocolo es similar a la conectividad inalámbrica de baja potencia de 2.4GHz que a menudo se implementa en ratones inalámbricos. Por lo tanto, el emparejamiento entre dispositivos es necesario antes de su comunicación. Una vez que se realiza el emparejamiento, la conexión es segura y de igual a igual, sin que sea necesario un apretón de manos. "

Mas información en el link:
https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/network/esp_now.html
ESP-NOW es un protocolo amplio y con muchas posibilidades, pero quiero mostrar una forma fácil de comunicar dos dispositivos y transmitir datos entre ellos, sin utilizar formas complejas.

El sketch que he preparado solo un dispositivo transmite (joystick) y otro recibe sus datos (vehículo). Pero ambos deben tener cosas comunes necesariamente, las cuales paso a describir.
.- Inicio de la librería ESP-NOW

```
//Iniciamos la libreria de ESP-NOW
#include <ESP8266WiFi.h>
extern "C" {
#include <espnow.h>
}
```

.- La estructura de datos a transmitir/recibir. No podemos definir las variables con longitud variable, sino de longitud fija, debido a cuando se transmiten todos los datos a la vez, el que recibe debe saber separar cada byte recibido y saber a que valor de variable asignar dichos bytes recibidos. Es como cuando se prepara un tren, con distintos vagones y la estación que los recibe debe saber cuantos y para que empresa deben ir. Quiero transmitir 5 datos a la vez, Si pulso el joystick, y los voltajes (motor Izquierdo y Derecho) y sentido (adelante/atrás) de cada motor del vehículo, que extraigo de la posición del mismo.

```
struct Data_ESP_NOW {
  uint8_t Pulsador;
  uint16_t VoltajeMI;
  uint16_t VoltajeMD;
  uint16_t SentidoMI;
  uint16_t SentidoMD;
};
```

.- Defino el tipo de función que realizará cada Wemos. Quizás debido a la falta de experiencia en el protocolo ESP-NOW, he tenido ciertos problemas cuando a uno lo defino como maestro y al otro como esclavo. Siempre me ha funcionado bien poniendo los dos como bidireccionales (Role=3)

```
  //0=sin función, 1=MAESTRO, 2=ESCLAVO y 3=MAESTRO+ESCLAVO
  esp_now_set_self_role(3);
```

.- Emparejamiento de los dispositivos. En el sketch del joystck debo poner la AP MAC de la Wemos del vehículo. En el sketch del vehículo, debo poner la AP MAC del joystick.

```
uint8_t mac_addr[6] = {0x86, 0xF3, 0xEB, 0xB7, 0x32, 0x09};
uint8_t role = 3;
uint8_t channel = 3;
//si la longitud es cero y la matriz vacia, no hay clave
uint8_t key[12] = {0x3A, 0x2B, 0x78, 0x03, 0x77, 0x58, 0x86, 0xF3, 0xEB, 0xB7, 0x30, 0xFA};
uint8_t key_len = sizeof(key);
esp_now_add_peer(mac_addr, role, channel, key, key_len);
```

AP MAC

.- Envío de datos al vehículo, figura siguiente. Primero hay que preparar esos vagones del tren que hay que enviar (data), con recuadro rojo. Después, hay que definir a quien lo envío (da), **que es la AP MAC de la Wemos del vehículo** y la longitud total del TREN. Una vez definidos estos datos anteriores, se envía el paquete de datos (cuadro verde).
Recuerda: Quiero transmitir 5 datos a la vez, Si pulso el joystick, y los voltajes (motor Izquierdo y Derecho) y sentido (adelante/atrás) de cada motor del vehículo.
Tras el envío, verifico que el vehículo ha recibido los datos correctamente (cuadro azul).

```
void transmision() {
  //***ENVIO DE LOS DATOS a uno ya emparejado previamente
  //Serial.print("Entro en transmision().  "); Serial.println(Comando);
  //***ENVIO DE LOS DATOS a uno ya emparejado previamente
  // Extructura del tren de datos a enviar
  Data_ESP_NOW TREN;
  TREN.Pulsador = Pulsador;
  TREN.VoltajeMI = VoltajeMI;
  TREN.VoltajeMD = VoltajeMD;
  TREN.SentidoMI = SentidoMI;
  TREN.SentidoMD = SentidoMD;
  uint8_t da[6] = {0x86, 0xF3, 0xEB, 0xB7, 0x32, 0x09};
  uint8_t data[sizeof(TREN)]; memcpy(data, &TREN, sizeof(TREN));
  uint8_t len = sizeof(data);
  esp_now_send(da, data, len); //envio de datos
  //***VERIFICACION DE LA RECEPCION CORRECTA DE LOS DATOS POR EL ESCLAVO***//
  esp_now_register_send_cb([](uint8_t* mac, uint8_t status) {
    //char MacTX[6];
    //sprintf(MacTX, "%02X:%02X:%02X:%02X:%02X:%02X", mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
    //Serial.print(". Enviado a ESP MAC: "); Serial.println(MacTX);
    //Serial.print(". Recepcion (0=OK - 1=ERROR):    "); Serial.println(status);
    TXgood = status;   // 0=OK
  });
}
```

.- Recepción de datos en el vehículo. Esta es la función que he usado en la Wemos del vehículo. Como se puede ver la pongo en modo de recepción (con respuesta, call back) y la data recibida la asigno a las variables (vagones del TREN) con la misma estructura utilizada en ambos:

```
void recepcion() {
  esp_now_register_recv_cb([](uint8_t *mac, uint8_t *data, uint8_t len ) {
    //char MACrecibida[6];
    //sprintf(MACrecibida, "%02X:%02X:%02X:%02X:%02X:%02X", mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
    //Serial.print("Recepcion desde ESP MAC: "); Serial.println(MACrecibida);
    Data_ESP_NOW TREN;
    memcpy(&TREN, data, sizeof(TREN));
    Pulsador = TREN.Pulsador;
    VoltajeMI = TREN.VoltajeMI;
    VoltajeMD = TREN.VoltajeMD;
    SentidoMI = TREN.SentidoMI;
    SentidoMD = TREN.SentidoMD;
  });  //Ojo con esos dos símbolos. Cierran la recepción
}
```

Y simplemente con lo anterior, puedo transmitir/recibir datos vía Wifi ESP-NOW de forma sencilla.

En el siguiente paso va a ser una descripción del sketch de Arduino del mando a distancia (joystick)

Arduino (Joystick)

.-Tras definir la librería de ESP-NOW, defino los pines que voy a utilizar de la Wemos:

```
// Pines de la placa
#define inA0 A0  //Entrada analogica del joystick (ambos potenciometros, Pot1 y Pot2)
#define inD0 16  // Entrada digital de la posicion del pulsador del joystick
#define outD1 5  // Salida digital para activar la lectura de Pot1
#define outD2 4  // Salida digital para activar la lectura de Pot2
#define outD5 14  // Salida digital para activar la lectura de Medida de bateria D4
#define outTXG 0 // Salida de led de TX Good D3
```

.- Defino las variables que usaré posteriormente:

```
// Definicion de variables. Las 5 primeras son las que transmito.
byte Pulsador = 0;  // Variable del valor del pulsador del joystick
int VoltajeMI = 0; // PWM hacia el L298N motor izqierdo
int VoltajeMD = 0; // PWM hacia el L298N motor derecho
int SentidoMI = 0; // Hacia donde gira el motor izquierdo (0=parado, 1=adelante, 2=atrás) al L298N
int SentidoMD = 0; // Hacia donde gira el motor derecho (0=parado, 1=adelante, 2=atrás) al L298N
int Pot1 = 0; // variable analogica donde se almacena el valor de posicion horizontal en mi circuito
int Pot2 = 0;  // variable analogica donde se almacena el valor de posicion vertical en mi circuito
int Bat = 0;  // variable analogica donde se almacena el valor del voltaje de la bateria
// Los valores de los potenciometros los paso a valores entre 0 y 1023 (Wemos) y con su direccion.Indican el mov:
int IZQ = 0;  // Izquierda
int DER = 0;  // Derecha
int DEL = 0;  // Hacia adelante
int ATR = 0;  // Hacia atras
byte TXgood = 0;  //Si transmite correctamente tiene valor 0
byte ContadorReposo = 0;  // Acumula e indica cuantos bucles de espera de movimiento, para analizar el estado de
byte Movimiento = 0; // 1 indica que el joystick se mueve
int MinMotor = 140; // PWM minimo en la salida para que arranquen los motores
int MaxMotor = 450; // PWM maximo en la salida para maximo voltaje en los motores. Junto con MinMotor, dependen
```

.- Ya en setup(), en la primera parte, defino como van a trabajar los pines de la Wemos y un valor inicial de los mismos. También verifico que el protocolo ESP-NOW esté inicializado bien. Y tras ello, defino el modo de trabajo y emparejamientos anteriormente comentados:

```
void setup() {
  //Serial.begin(500000);
  pinMode(inD0, INPUT);
  pinMode(outD1, OUTPUT); digitalWrite(outD1, LOW); // Apago las lecturas en los optoacopladores
  pinMode(outD2, OUTPUT); digitalWrite(outD2, LOW);
  pinMode(outD5, OUTPUT); digitalWrite(outD5, LOW);
  pinMode (outTXG, OUTPUT); digitalWrite(outTXG, LOW);  // Cuando transmite OK, se enciende

  //  Inicia protocolo ESP-NOW si no lo está
  if (esp_now_init() != 0) {
    //Serial.println("Protocolo ESP-NOW no inicializado...");
    ESP.restart();
    delay(10);
  }
  else {
    //Serial.println("Protocolo ESP-NOW INICIALIZADO --OK--...");
  }
```

.- Inicio el loop() con un retardo que nos marca el número de transmisiones o lecturas del joystick que quiero hacer por segundo (figura siguiente). He puesto 60 msg, con lo que realizo unas 15 lecturas por segundo mas o menos. Después leo el estado del pulsador de emergencia. Si se pulsa, pongo a cero los valores de los motores, transmito y establezco un retardo donde no responde a nada hasta que pase ese tiempo (en mi caso de 5 segundos, delay(5000);).

```
void loop() {
  delay(60);  // Retardo entre tomas de datos de ambos potenciometros (sincronizar en el esclavo mas o menos)

  VoltajeMI = 0; VoltajeMD = 0; SentidoMI = 0; SentidoMD = 0; //Los valores  los pongo a 0

  Pulsador = digitalRead(inD0);  // Lee estado del Pulsador, por si hay emergencia
  if (Pulsador == 1) {  // Si se pulsa, envia el estado y el esclavo paraliza los motores durante 5 segundos (se puede varia
    transmision();  // Transmite que se ha pulsado "Emergencia"
    Pulsador = 0;
    if (TXgood == 0) { // Si se pulsa y ha transmitido ok, se mantiene encendido durante el retardo y bloquea el joystick
      digitalWrite(outTXG, HIGH);  //Se enciende el led si la transmision es correcta
    }
    else {
      digitalWrite(outTXG, LOW);
    }
    delay(5000);   // Fijo el retardo que quiero en estado de "Parada de emergencia" (en transmisor y receptor igual)
  }
```

.- El resto del loop(), son las llamadas a las funciones que utilizo, que posteriormente explicaré.

```
  //      Si no está pulsador el switch, lee los potenciometros
  leePots();  // Realiza la lectura del potenciometro horizontal/vertical del joystick y del estado de la bateria (Pot1, Pot2 y Bat)

  ajustePots(); // Los valores de los Potenciometros horizontal/vertical, los transforma en valores hacia adelante/atras y derecha/izquierda (DEL, ATR, IZQ y DER)

  dirMot();  //Los valores DEL, ATR, IZQ y DER, los transformo en voltajes PWM de motores y el sentido de giro (VoltajeMI, VoltajeMD, SentidoMI y SentidoMD)

  // SI SE DESVIA EL VEHICULO, CORRIJO LA TRAYECTORIA AQUI antes de transmitir
  VoltajeMD = VoltajeMD + 50;  // en recto, se me desvia hacia la derecha, por lo que aumento el voltaje del motor derecho.
  transmision();

  if ((TXgood == 0) && (Movimiento == 1)) { //si la transmision ha sido correcta y hay movimiento
    digitalWrite(outTXG, HIGH);  //Se enciende el led
    ContadorReposo = 0;
    //Serial.print("Transmision correcta y Movimiento: TXgood=0...."); Serial.println(TXgood);
  }
  else {
    digitalWrite(outTXG, LOW);
    //Serial.print("ERROR al Transmitir o sin Movimiento: TXgood=1....."); Serial.println(TXgood);
  }

  ControlBat();
  // FIN de LOOP

}
```

.- Leo el estado de los potenciómetros y de la batería. leePots( ); . Los retardos (delay) que pongo de 5msg son para que las lecturas en los optoacopladores sean precisas. Hay que tener en cuenta que desde que se activa el led, tarda unos microsegundos (unos 10) en estabilizar la salida, así que le pongo 5 msg para que las lecturas sean mas correctas. Se podría bajar este retardo perfectamente.

```
void leePots() {
  // Activa D1 y lee el valor de P1 del Joystick (Izquierda/derecha en mi circuito)
  digitalWrite(outD2, LOW);  //Asegura que solo lee Pot1
  digitalWrite(outD5, LOW);
  digitalWrite(outD1, HIGH);
  //Serial.println("Activado D1 ");
  delay(5);  // Retardo que necesita el optoacoplador para estabilizar la salida
  Pot1 = analogRead(inA0);
  digitalWrite(outD1, LOW);  // Apaga la lectura
  delay(5);

  // Activa D2 y lee el valor de P2 del Joystick (Delante/atras en mi circuito)
  digitalWrite(outD1, LOW);  //Asegura que solo lea Pot2
  digitalWrite(outD5, LOW);
  digitalWrite(outD2, HIGH);
  //Serial.println("Activado D2 ");
  delay(5);  // lectura estabilizada en el optoacoplador
  Pot2 = analogRead(inA0);
  digitalWrite(outD2, LOW);  // Apaga la lectura
  //delay(5);

  // Activa D4 y lee el valor del voltaje de la bateria
  digitalWrite(outD1, LOW);  //Asegura que solo lea el nivel de la bateria
  digitalWrite(outD2, LOW);
  digitalWrite(outD5, HIGH);
  //Serial.println("Activado D5 ");
  delay(5);  // lectura estabilizada en el optoacoplador
  Bat = analogRead(inA0);
  digitalWrite(outD5, LOW);  // Apaga la lectura
  //delay(5);
}
```

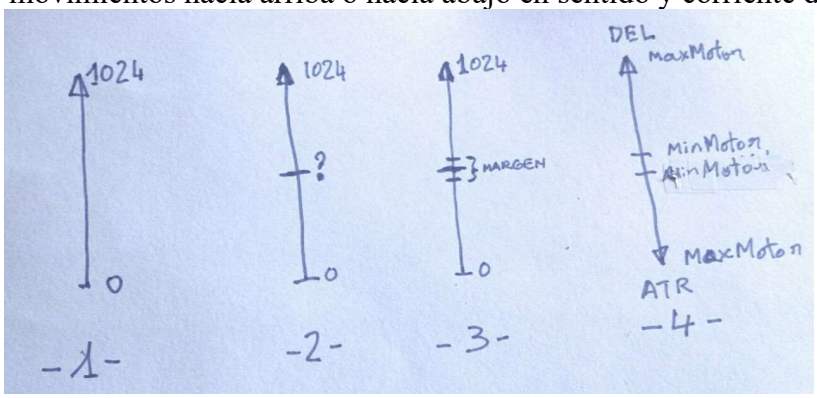
.- Una vez leídos los potenciómetros y el estado de la batería, hay que transformar el movimiento del joystick en sentido y corriente hacia los motores. Si analizamos el potenciómetro vertical, por ejemplo, los pasos están mostrados en la figura siguiente.

1.- El valor total en el movimiento (mínimo, reposo, máximo) está entre 0 y 1024.

2.- Averiguar cual es el punto medio del mismo (reposo de la palanca). Ver leePot();

3.- Establecer un margen para que no se mueva el vehículo con ligeros movimientos o que no afecten las fluctuaciones eléctricas.

4.- Convertir los movimientos hacia arriba o hacia abajo en sentido y corriente de los motores.



Los pasos 2 a 4 los realizo en ajustePots();.

```
void ajustePots() {
  // MinMotor es el valor minimo en el L298N para que la señal PWM haga mover los motores. MaxMotor es el máximo voltaje de los motores y depende de ellos y de la ateria usada
  // Transforma el valor del potenciómetro Pot1 (izquierda/Derecha) en direcciones
  // Dar un margen de valores en el centro del mismo, para evitar fluctuaciones en reposo del joystick.
  // En las pruebas, en el centro, Pot1 vale 546. Centro entre 556 y 536 (margen). Para cada joystick, estos valores pueden variar. Ajustar tras las pruebas con el joystick en reposo.
  IZQ = 0; // Borra los anteriores valores
  DER = 0;
  int temp = 0;
  //Serial.print("Pot_1:   "); Serial.println(Pot1);
  if (Pot1 >= 556) {  // Asegura el centro del mismo, en reposo, mi joystick da 546. Fijo un margen de 10 izquierda y derecha
    IZQ = map(Pot1, 556, 1024, MinMotor, MaxMotor);  // Los dos ultimos dependen de los motores y bateria. Uno es el minimo para que puedan arrancar y el otro marca el voltaje maximo
    // Tras este mapeo, IZQ toma un valor entre MinMotor y MaxMotor
  }
  if (Pot1 <= 536) {
    temp = 1024 - Pot1;  //Invierto el valor 1024-536(menor)=488. Al dismnuir el valor de Pot1, aumenta el valor de hacia la derecha
    DER = map(temp, 488, 1024, MinMotor, MaxMotor);
  }

  // Transforma el valor del potenciómetro Pot2 en avance/retroceso de los motores
  // Dar un margen de valores en el centro del mismo, para evitar fluctuaciones en reposo del joystick
  // En las pruebas en reposo mi joystick, da Pot2 529. Centro entre 539 y 519 (margen)
  // Pot2 nos marca el sentido de movimiento
  DEL = 0; // Borra los anteriores valores
  ATR = 0;
  temp = 0;
  //Serial.print("Pot_2:   "); Serial.println(Pot2);
  if (Pot2 >= 539) {
    DEL = map(Pot2, 539, 1024, MinMotor, MaxMotor);  // Los dos ultimos dependen de los motores y bateria. Uno es el minimo para que puedan arrancar y el otro marca el voltaje maximo
  }
  if (Pot2 <= 519) {  // Invierto el valor del joystick. 1024-519(menor)=505
    temp = 1024 - Pot2;
    ATR = map(temp, 505, 1024, MinMotor, MaxMotor);   //
  }
}
```

.- Partimos del hecho de que un dispositivo de dos motores, sin eje de dirección, necesita unos valores de sentido y voltaje hacia los mismos. La conversión de hacia adelante/atrás y hacia la izquierda/derecha en sentido/voltaje lo realizo en dirMot(), teniendo en cuenta las 3 direcciones hacia adelante izquierda/frontal/derecha, lo mismo hacia atrás e incorporo el giro sobre sí mismo. Cuando va hacia adelante y giro, lo que hago es reducir el voltaje de la rueda a la que giro, proporcionalmente al movimiento del joystick y evitando los valores negativos, por lo tanto, el valor de reducción nunca puede ser menor que el valor de avance (como mucho, para el motor). De ahí el uso de la variable de giro (VariableGiro).

```
void dirMot() {
  /*
     Los motores giran según los distintos casos siguientes.
     .-Parado
     .-Giro sobre si mismo (2 sentidos)
     .-Avance (sin giro, con giro derecha y con giro izquierda
     .-Retroceso (sin giro, con giro derecha y con giro izquierda
  */

  int VariableGiro = 0;

  if ((DEL == 0) && (ATR == 0) && (IZQ == 0) && (DER == 0)) { // Si el joystick está centrado (todos == 0)
    Movimiento = 0;
    VoltajeMI = 0; // PWM hacia el motor izqierdo
    VoltajeMD = 0; // PWM hacia el motor derecho
    SentidoMI = 0; // Hacia donde gira el motor izquierdo (0=parado, 1=adelante, 2=atrás)
    SentidoMD = 0;  // Hacia donde gira el motor derecho (0=parado, 1=adelante, 2=atrás)
    //Serial.println("Parado");
    return;
  }
  // ***** GIRO SOBRE SI MISMO *****
  // Solo cuando el joystick está en pocicion centrado verticalmente y DER/IZQ tienen un valor positivo uno de ellos
  if ((DEL == 0) && (ATR == 0)) {  //solo giros de izquierda o derecha
    //Serial.println("Giro sobre si mismo");
    if (DER > 0) { // Giro sobre si mismo a la derecha
      Movimiento = 3;
      SentidoMI = 1;  //motor izquierdo hacia adelante
      SentidoMD = 2;  // motor derecho hacia atrás
      VoltajeMI = 160;  // ambos motores a una velocidad determinada, pero en sentido contrario
      VoltajeMD = 160;  // Estos valores, siempre mayores a la velocidad minima de comienzo de movimiento
      return;
      //Serial.println("Giro sobre si mismo a la derecha");
    }
    if (IZQ > 0) { // Giro sobre si mismo a la izquierda
      Movimiento = 3;
      SentidoMI = 2;  // motor izquierdo hacia atrás
      SentidoMD = 1;  // motor derecho hacia adelante
      VoltajeMI = 160;  // ambos motores a una velocidad determinada, pero en sentido contrario
      VoltajeMD = 160;  // Estos valores, siempre mayores a la velocidad minima de comienzo de movimiento
      return;
      //Serial.println("Giro sobre si mismo a la izquierda");
    }
  }

  // Para los avances con giro , al motor que marca el giro le resto un valor proporcional al movimiento del joystick, pero siempre menor que el avance/retroceso
  // Si el giro es muy brusco, restar un numero a las variables ultimas del MAP, ejemplo (DEL-10). Asi el motor que reduce nunca va a cero auque el joystick lo tengamos al tope de direcci
  // Ejemplo: Si DEL=300 y giro el joystick a la izquierda, el motor derecho va a 300 y el izquierdo va DE 300 A 0 (avanza menos o para), dependiendo de su posicion.
  if ((DEL > 0) && (IZQ > 0)) {
    VariableGiro = map(IZQ, MinMotor, MaxMotor, 0, DEL);
  }
  if ((DEL > 0) && (DER > 0)) {
    VariableGiro = map(DER, MinMotor, MaxMotor, 0, DEL);
  }
  if ((ATR > 0) && (IZQ > 0)) {
    VariableGiro = map(IZQ, MinMotor, MaxMotor, 0, ATR);
  }
  if ((ATR > 0) && (DER > 0)) {
    VariableGiro = map(DER, MinMotor, MaxMotor, 0, ATR);
  }

  // ***** AVANZA*****
  if (DEL > 0) {  // avance hacia adelante.Tres casos. Hacia la derecha, centro o izquierda
    //Serial.println("Marcha Adelante");
    if ((DER == 0) && (IZQ == 0)) { // Avance sin giro
      SentidoMI = 1;  // motor izquierdo hacia adelante
      SentidoMD = 1;  // motor derecho hacia adelante
      VoltajeMI = DEL; // ambos motores a la misma velocidad o modificar la trayectoria recta restando algo a uno de los valores en loop() antes de transmitir
      VoltajeMD = DEL;
      Movimiento = 3;
      return;
      //Serial.println("Adelante sin giro");
    }
    if (DER > 0) {  // avance con giro a la derecha
      SentidoMI = 1;  // motor izquierdo hacia adelante
      SentidoMD = 1;  // motor derecho hacia adelante
      VoltajeMI = DEL;                  // El motor izquierdo avanza a la velocidad de hacia adelante
      VoltajeMD = DEL - VariableGiro;   //El motor derecho gira menos, dependiendo del joystick hacia la derecha.
      Movimiento = 3;
      return;
      //Serial.println("Adelante-Derecha");
    }
    if (IZQ > 0) {  // avance con giro a la izquierda
      SentidoMI = 1;  // motor izquierdo hacia adelante
      SentidoMD = 1;  // motor derecho hacia adelante
      VoltajeMI = DEL - VariableGiro;   //El motor izqquierdo gira menos, dependiendo del joystick hacia la izquierda. Disminuir el ultimo numero si queremos que gire menos
      VoltajeMD = DEL;                  //El motor derecho avanza a la velocidad de hacia adelante
      Movimiento = 3;
      return;
      //Serial.println("Adelante-Izquierda");
    }
  }

  // ***** RETROCEDE *****
  if (ATR > 0) {  // avance hacia atras. Tres casos. Hacia la derecha, centro o izquierda
    //Serial.println("Marcha Atras");
    if ((DER == 0) && (IZQ == 0)) { // Avance sin giro
      SentidoMI = 2;  // motor izquierdo hacia atrás
      SentidoMD = 2;  // motor derecho hacia atras
      VoltajeMI = ATR; // ambos motores a la misma velocidad o modificar la trayectoria recta restando algo a uno de los valores en loop() antes de transmitir
      VoltajeMD = ATR;
      Movimiento = 4;
      return;
      //Serial.println("Atras sin giro");
    }
    if (DER > 0) {  // retrocede con giro a la derecha
      SentidoMI = 2;  // motor izquierdo hacia atras
      SentidoMD = 2;  // motor derecho hacia atras
      VoltajeMI = ATR;                  // El motor izquierdo avanza lo normal
      VoltajeMD = ATR - VariableGiro ;  //El motor derecho gira menos, dependiendo del joystick hacia la derecha. Disminuir el ultimo numero si queremos que gire menos
      Movimiento = 4;
      return;
      //Serial.println("Atras-Derecha");
    }
    if (IZQ > 0) {  // avance con giro a la izquierda
      SentidoMI = 2;  // motor izquierdo hacia atras
      SentidoMD = 2;  // motor derecho hacia atras
      VoltajeMI = ATR - VariableGiro;   //El motor izqquierdo gira menos, dependiendo del joystick hacia la izquierda. Disminuir el ultimo numero si queremos que gire menos
      VoltajeMD = ATR;                  // El motor derecho avanza lo normal
      Movimiento = 4;
      return;
      //Serial.println("Atras-Izquierda");
    }
  }
}
```

.- Por último, el control del estado de la batería. Cuando el joystick está en reposo, o no ha podido transmitir, incremento un contador. Si alcanza un valor deseado (50 veces), analizo el estado de la batería y hago parpadear el led (1 parpadeo=baja, 2 parpadeos=muy baja)

```
void ControlBat() {
  if (Movimiento = 0) {
    ContadorReposo = ContadorReposo + 1;
    digitalWrite(outTXG, LOW);
    //Serial.print("Contador ");  Serial.println(ContadorReposo);
  }
  else {
    //digitalWrite(outTXG, HIGH);
    ContadorReposo = 0;
  }
  if (ContadorReposo >= 50) {
    // 1024 equivale a 3.3v.
    // Con bateria de 3,7v (Normal cargada a tope=4v) y el divisor de voltaje, el valor máximo es 2v
    // Con la bateria cargada (4v) el valor leido tras el divisor es de 620 (2v)
    // Con la bateria baja (3.5v) el valor leido tras el divisor es de 543 (1.75v)
    // Con la bateria MUY baja (3.4v) el valor leido tras el divisor es de 527 (1.7v)
    //Serial.print("Valor de bateria ");  Serial.println(Bat);
    //Serial.println("");
    if (Bat <= 543 && Bat > 527) { //Bateria baja. 1 parapadeo
      digitalWrite(outTXG, HIGH);  // enciende el led de TXGood
      delay(100);
      digitalWrite(outTXG, LOW);  // apaga el led de TXGood
      delay(100);
    }
    if (Bat < 527) { //Bateria MUY baja. 3 parpadeo
      digitalWrite(outTXG, HIGH);  // enciende el led de TXGood
      delay(100);
      digitalWrite(outTXG, LOW);  // apaga el led de TXGood
      delay(100);
      digitalWrite(outTXG, HIGH);  // enciende el led de TXGood
      delay(100);
      digitalWrite(outTXG, LOW);  // apaga el led de TXGood
      delay(100);
    }
    ContadorReposo = 0;
  }
}
```

Una vez comentado el sketch del joystick para Arduino, pasamos a ver el sketch del vehículo.

Arduino (Vehículo)

Sobre la parte correspondiente a las comunicaciones (ESP-NOW) con el joystick, ya se comentaron anteriormente, por lo que analizo el resto. Hay que tener en cuenta de que lo he simplificado bastante, para que si hay que hacer modificaciones, se trabaja mejor modificando el mando a distancia que a tener que poner el vehículo en la mesa y conectarlo al ordenador. Por ello, me limito a recoger los datos de movimiento y pasarlos al L298N para que se muevan los motores. Priorizo la recepción del pulsador de emergencia y en los tiempos sin movimiento, analizo el estado de la batería.

.- Pines de entrada salida de la placa Wemos y Variables usadas:

```
// Pines de Salida hacia el L298N Control de Motores
#define pinENA 5   //D1   PWM Pasos de motor izquierdo
#define pinIN1 4   //D2   Motor Izq hacia a delante
#define pinIN2 2   //D4   Motor Izq hacia atras  //D3 da problemas y se mantiene siempre HIGH
#define pinENB 16  //D0  PWM Pasos de motor derecho
#define pinIN3 14  //D5  Motor Der hacia a delante
#define pinIN4 12  //D6  Motor Der hacia atras
#define pinEME 13  //D7  Emergencia
#define pinRXB 15  //D8  RX-Bad
#define pinBat 0   //A0  Nivel de bateria

// Valores iniciales de Variables
int Pulsador = LOW;  // Variable del valor del pulsador del joystick
int RXgood = 0;  //Si transmite correctamente
int Bateria = 0;   // Memoriza el estado de la bateria. Muestra de entrada de 12v con un divisor 47k/4k7
int VoltajeMI = 0; // PWM hacia el motor izqierdo
int VoltajeMD = 0; // PWM hacia el motor derecho
int SentidoMI = 0; // Hacia donde gira el motor izquierdo (0=parado, 1=adelante, 2=atrás)
int SentidoMD = 0; // Hacia donde gira el motor derecho (0=parado, 1=adelante, 2=atrás)
int ContadorInactivo = 0; // Controla cuando pasa un tiempo sin recibir datos, para medir el nivel de bateria y hacer parpadear el led
```

.- ya en el setup() inicio los pines y su estado inicial. El resto de setup es sobre ESP-NOW:

```
void setup() {
  Serial.begin(500000);

  pinMode(pinENA, OUTPUT);  pinMode(pinIN1, OUTPUT);  pinMode(pinIN2, OUTPUT);  // motor izquierdo en L298N
  digitalWrite(pinIN1, LOW);  digitalWrite(pinIN2, LOW);  analogWrite(pinENA, 0);  //Paro motor izquierdo

  pinMode(pinENB, OUTPUT);  pinMode(pinIN3, OUTPUT);  pinMode(pinIN4, OUTPUT);  // motor derecho en L298N
  digitalWrite(pinIN3, LOW);  digitalWrite(pinIN4, LOW);  analogWrite(pinENB, 0);  //Paro motor derecho

  pinMode(pinEME, OUTPUT);  pinMode(pinRXB, OUTPUT);  pinMode(pinBat, INPUT);  //Leds y entrada analogica de la bateria.
  digitalWrite(pinRXB, LOW);  // Se enciende si recibe datos ok.
  digitalWrite(pinEME, LOW);  // Se enciende al pulsar Emergencia o parpadea si está baja la bateria

  //***INICIALIZACION DEL PROTOCOLO ESP-NOW si no lo esta***  (ESP-NOW)***//
  if (esp_now_init() != 0) {
```

.- En loop(), aparte de mirar el estado de la batería, mando ejecutar dos funciones, una comentada ya al hablar del ESP-NOW, recepción() y la otra realiza el manejo del L298N con los datos recibidos. Por supuesto, lo primero es analizar una posible emergencia y parar el vehículo. Primero establezco un pequeño retardo en las comunicaciones, para sincronizar el receptor mas o menos con el transmisor. Ejecuto la función de recepción() y analizo si se ha pulsado “Emergencia” para proceder a la inmovilización. Si no recibo datos o movimiento de ninguno de los motores, los paro también mediante el envío de datos a la función writeL298N(). Si no hay datos, incremento un contador para revisión de la batería. Si hay datos recibidos, enciendo el led de comunicaciones y por supuesto, los mando a la función writeL298N() para que se mueva el motor según dichos datos.

```
void loop() {

  delay(80);  //Si se baja, no controla bien el status de comunicaciones

  recepcion();  //Recoge los datos de la red

  if (Pulsador == 1) {
    VoltajeMI = 0; // PWM hacia el motor izqierdo
    VoltajeMD = 0; // PWM hacia el motor derecho
    SentidoMI = 0; // Hacia donde gira el motor izquierdo (0=parado, 1=adelante, 2=atrás)
    SentidoMD = 0;  // Hacia donde gira el motor derecho (0=parado, 1=adelante, 2=atrás)
    writeL298N();  // Con los valores anteriores, para los motores
    digitalWrite(pinEME, HIGH);
    //Serial.println("Parada de Emergencia. Esperar 5 segundos");
    delay(5000);   // RETARDO EN CASO DE PULSAR "EMERGENCIA"
    Pulsador = 0;
    digitalWrite(pinEME, LOW);
  }

  if ((SentidoMI == 0) && (SentidoMD == 0)) { // Cuando el movimiento recibido es cero o no recibe datos
    //Serial.println("***FALLO EN RECEPCION o sin movimiento***");
    ContadorInactivo = ContadorInactivo + 1;  // Si no recibo movimiento, incremento un contador
    digitalWrite(pinRXB, LOW);  //Apago el LED y paro motores
    VoltajeMI = 0; // PWM hacia el motor izqierdo
    VoltajeMD = 0; // PWM hacia el motor derecho
    SentidoMI = 0; // Hacia donde gira el motor izquierdo (0=parado, 1=adelante, 2=atrás)
    SentidoMD = 0; // Hacia donde gira el motor derecho (0=parado, 1=adelante, 2=atrás)
```

```
    writeL298N();  // Con los datos anteriores, paro los motores

  }
  else {
    ContadorInactivo = 0;
    digitalWrite(pinRXB, HIGH);  //Si hay recepcion de datos, enciende el led
  }

  if (ContadorInactivo >= 20) {  // Si está en reposo, controlo el estado de la bateria. Aumentar el numero si queremos que haga menos controles de la misma.
    // en A0, 1024 equivale a un voltaje de 3.3v.
    //Tras el divisor de voltaje de 11v, estando ok, medimos 1v, equivalente a 310 en A0
    Bateria = analogRead(pinBat);
    if (Bateria <= 153) { // "Bateria MUY baja" (5,5v) Si no hay transmisión, aprovecho para controlar el estado de la bateria
      //Serial.println("Bateria MUY baja");   //  3 parpadeos cortos del led de Emergencia
      digitalWrite(pinEME, HIGH); delay(50); digitalWrite(pinEME, LOW); delay(50);
      digitalWrite(pinEME, HIGH); delay(50); digitalWrite(pinEME, LOW); delay(50);
      digitalWrite(pinEME, HIGH); delay(50); digitalWrite(pinEME, LOW); delay(50);
      ContadorInactivo = 0;
    }
    if (Bateria > 153 && Bateria <= 195) {  // "Bateria baja", entre 7v y 5.5v
      //Serial.println("Bateria baja");  // 1 parpadeo del led de Emergencia
      digitalWrite(pinEME, HIGH); delay(100); digitalWrite(pinEME, LOW); delay(100);
      ContadorInactivo = 0;
    }
    else {
      //Serial.println("Bateria OK");
    }

  }

  writeL298N();

  //FIN de loop()
}
```

.- Función writeL298N()

Si recordais la tabla del L298N, simplemente es escribir dichos valores con los datos recibidos

```
void writeL298N() {
  //Serial.println("Movimiento de Motores");
  // Motor Izquierdo   =   ENA, IN1, IN2
  //Serial.println("Muevo Motores "); Serial.println("");
  switch (SentidoMI) {  //Izquierda
    case 0:  // Parado
      digitalWrite(pinIN1, LOW);
      digitalWrite(pinIN2, LOW);
      analogWrite(pinENA, 0);
      break;
    case 1:  // Adelante
      digitalWrite(pinIN1, HIGH);
      digitalWrite(pinIN2, LOW);
      analogWrite(pinENA, VoltajeMI);
      //Serial.print("Adelante Motor Izquierdo "); Serial.println(VoltajeMI);
      break;
    case 2:  // Atrás
      digitalWrite(pinIN1, LOW);
      digitalWrite(pinIN2, HIGH);
      analogWrite(pinENA, VoltajeMI);
      //Serial.print("Atras Motor Izquierdo "); Serial.println(VoltajeMI);
      break;
  }
  // Motor Derecho  =   ENB, IN3, IN4
  switch (SentidoMD) {  //Izquierda
    case 0:  // Parado
      digitalWrite(pinIN3, LOW);
      digitalWrite(pinIN4, LOW);
      analogWrite(pinENB, 0);
      break;
    case 1:  // Adelante
      digitalWrite(pinIN3, HIGH);
      digitalWrite(pinIN4, LOW);
      analogWrite(pinENB, VoltajeMD);
      //Serial.print("Adelante Motor Derecho "); Serial.println(VoltajeMD);
      break;
    case 2:  // Atrás
      digitalWrite(pinIN3, LOW);
      digitalWrite(pinIN4, HIGH);
      analogWrite(pinENB, VoltajeMD);
      //Serial.print("Atras Motor Derecho "); Serial.println(VoltajeMD);
      break;
  }
  //Serial.print("Motor Izquierdo "); Serial.println(VoltajeMI);
  //Serial.print("Motor Derecho "); Serial.println(VoltajeMD);
}
```

Ésto es todo. No es mi intención ganar concursos, sino aclarar conceptos. Si UNA persona agradece este trabajo, le sirve para adquirir un conocimiento y después desarrollar alguna idea propia, me conformo. Si uno lo implementa en una silla de ruedas y hace mas confortable la vida a una persona, me haría mucha ilusión.

Un saludo:
Miguel A.