**ESCUELA DE EDUCACIÓN TÉCNICO PROFESIONAL**
**Y SECUNDARIA ORIENTADA**
**PARTICULAR INCORPORADA Nº8013**
**“SAN JOSÉ”**

# PROYECTO Y DISEÑO ELECTRÓNICO

**PRE INFORME**

**AUTO RC**

**APELLIDO, Nombre**: OSIA, Nahuel
VACCARI, Felipe

**Docentes**: GRACIA, Martin
MARTINEZ, Mauricio

**Curso:** 6to 1era

# Índice

# Introducción

A continuación, se verá en detalle el progreso y nuestra posición actual en lo que refiere al proyecto. Este proyecto llamado VONF, consta de un explorador con comunicación inalámbrica con un protocolo diseñado desde cero, con alta eficiencia. Dicha estructura del explorador está diseñada y llevada a cabo completamente con impresión 3D, tanto del auto como el control. A diferencia de la mayoría de los autos, este cuenta con una sensibilidad y precisión absoluta, debido al uso de los conversores ADC, y la robustez del protocolo de codificación. Cuenta con sistemas de iluminación trasera, delantera, lector de batería, leds indicadores de comunicación, de encendido, y un agregado de potencia extra al motor al presionar una combinación de botones (turbo programado).

## Gastos

En lo que respecta gastos, es muy pero muy volátil, o variable, esto debido a la situación económica del país, que entre otras cosas, radica en no saber si se pueden traer componentes de afuera, y su subida de precio en pesos. Esta lista no es definitiva ni mucho menos, ya que estamos en proceso de búsqueda de estos componentes extrayendolos de proyectos viejos y tal, pero un estimativo bruto, es el siguiente:

| Gastos aproximados | |
|---|---|
| Diseño 3D | 3270 |
| Módulo 433MHz | 1280 |
| Tornillería | 700 |
| Pic 16F886 | 6800 |
| Pic 16F628A | 2000 |
| Motor CC | 15000 |
| Servo motor | 4000 |
| Joysticks (2) | 800 |
| Baterias 12v 3ah | 20000 |
| Modulo de carga | 800 |
| Placa universal | 2000 |
| Rodamientos | 9000 |
| LCD 16x2 | 2200 |
| Drive motor cc | 1800 |
| Total | 69650 |

Entre los componentes no se tienen en cuenta resistencias, cables y tal. Asimismo, existe la posibilidad de expandir el proyecto y añadir ciertos sensores. Pero esto no está seguro, por eso no sacaremos cuentas de más, todo dependerá del transcurso constructivo del proyecto.

# Impresión 3D

**Chasis:**

El utilizar materiales convencionales como lo son la madera, la chapa, el acrílico o materiales reciclados, nos dificultaría muchísimo la etapa de diseño y armado del auto. Además, podría llegar a ser más costosa y nos veríamos limitados creativamente.

Por esto, decidimos utilizar cómo principal herramienta una impresora 3D. Lo que nos permitiría crear y modificar a gusto, según nuestra imaginación, el vehículo que tanto deseamos.

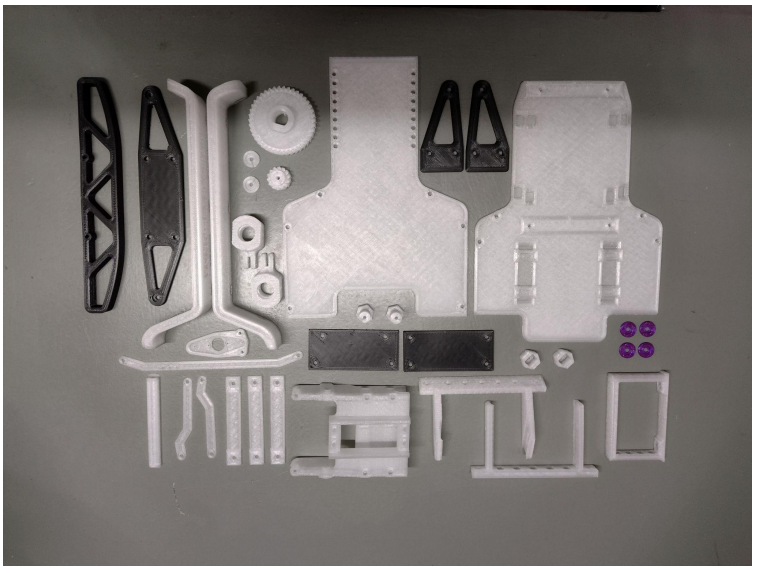
El hacer un auto casi al 100% imprimible en 3D trae muchas ventajas, cómo lo son la posibilidad de mejorarlo o modificarlo en un futuro, además la facilidad con la que podríamos cambiar piezas en caso de roturas y la enorme reducción de gastos.

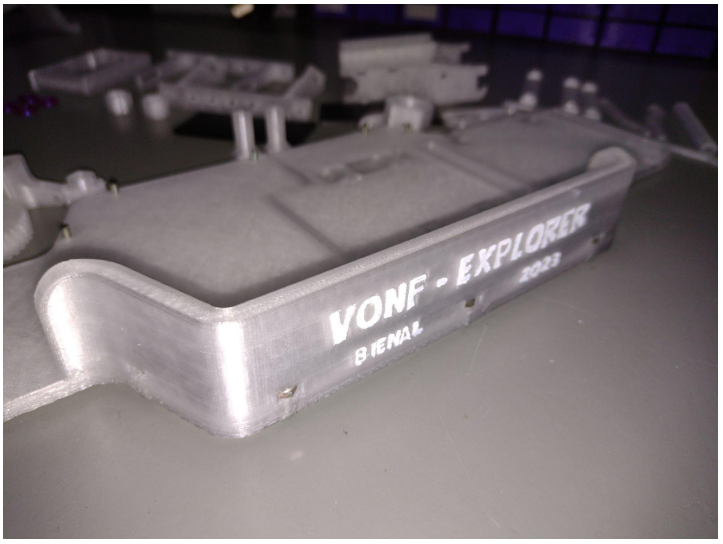
Tras una larga búsqueda e investigación acerca de diferentes modelos, decidimos utilizar como referencia un modelo base realizado por un profesional, el cuál usamos de guía para modificarlo a gusto y según nuestros requerimientos, ya que como mencionamos anteriormente, nuestro objetivo es poder agregarle sensores en un futuro.

En el caso del chasis, optamos por utilizar como material principal el PETG, el cuál está compuesto por PET (material utilizado para la fabricación de botellas plásticas) y por Glicol (un compuesto que se le añade al material para disminuir su punto de fusión y por lo tanto facilitar su extrusión). Este material posee mejores características técnicas y mecánicas en comparación al PLA (material más común en la impresión 3D), lo que se traduce en un mejor desempeño y mayor durabilidad del auto.

Además de utilizar PETG, aprovechamos las características técnicas particulares del TPU (material flexible) para imprimir el sistema de suspensión, el paragolpes y los neumáticos.

Para el caso de las piezas impresas en PETG, se requirieron 400g de material y cerca de unas 45 horas de impresión. En el caso de las piezas en TPU, alrededor de 120g y 20 horas de impresión. Esto se traduce a un total de $2500 (sin considerar el gasto de electricidad), lo que nos pareció un precio nada exorbitante.




**Ruedas:**

En un comienzo pensamos en la posibilidad de comprar ruedas ya armadas, pero debido al excesivo precio de cada una decidimos imprimirlas nosotros mismos. Si bien unas ruedas compradas poseen un mayor agarre, lo que se traduce en mejor tracción, consideramos que la tracción de nuestros neumáticos va a ser suficiente, ya que buscamos que sea un auto divertido de conducir.

Las mismas están impresas en PETG y los neumáticos en TPU. A las ruedas traseras las decidimos realizar un poco más anchas, para darle un aspecto más robusto al vehículo.

## Tornillería:

En el caso de la tornillería del auto, se utilizará únicamente tornillos de métrica 3. El vehículo lleva alrededor de 70 tornillos y 30 tuercas, que debido a la falta de disponibilidad de diferentes medidas, tuvimos que cortar con ayuda de una amoladora el total de los tornillos. Para esto, se tomó la medida que necesitábamos y se los colocó todos juntos en una chapa perforada. Luego realizamos el corte.

## Rodamientos:

En este apartado, se nos presentaron y estamos teniendo bastantes dificultades. Debido a la situación actual del país, no conseguimos stock de cierta medida específica de rodamientos que necesitamos para la transmisión del auto y para las ruedas. Si bien, en muy pocos lugares tienen stock, el precio no nos pareció nada acorde.

Utilizamos dos medidas de rodamientos:

Para el eje trasero (x2)

6701-2RS, 12x18x4mm

Para las ruedas delanteras (x4, 2 por rueda)

MR128-2RS, 8x12x3,5mm

## Control RC

Esta es nuestra actual etapa en desarrollo. Al empezar el proyecto, comprendimos que su complejidad se daba principalmente en la comunicación inalámbrica. Esto, debido a que hay que conocer los protocolos de comunicación, codificación, y así mismo tener en cuenta magnitudes físicas; la distancia, el ruido, el tamaño del dato a enviar, etc. Por eso mismo, vamos a fraccionar esta sección en diferentes subsecciones.

### ¿Qué se quiere transmitir?

Hay un mínimo de instrucciones a transmitir; Izquierda, derecha, adelante, atrás. Sin embargo, no queríamos que estas instrucciones cuenten tan solo con dos estados (ON-OFF), es por eso que decidimos asignar un rango de valores que definen la posición exacta (Derecha-Izquierda) y la velocidad del motor (Adelante-Atrás). Para elegir los valores de este rango, elegimos usar dos joysticks, los cuales tienen dos preset cada uno, (X e Y) y usamos uno de cada joystick, de esta forma un joystick define solo Izquierda o derecha (X) y el otro adelante y atrás (Y). Para hacer compatible el ingreso de información analógica a un Pic, es necesario un conversor ADC (Analogic to digital converter). Siendo así, nuestra primera tarea fue encontrar un pic con esta característica, la de tener dicho periférico. También podríamos haber comprado un modulo, pero sería más fácil programar por software, y en cuestión de precios nos convenía un pic más potente. Al conseguir un pic que cumplía con nuestros requerimientos, comenzamos uno de los primeros programas de práctica y ensayo:

```
ADCON0 = 0b10000001;
ADCON1 = 0b10000000;
unsigned int pwm, pot;
unsigned int resultado;

while (1){
    ADCON0bits.GO_nDONE=1;
    while(ADCON0bits.GO_nDONE==1);
    pot = ADRESH << 8;
    pot&= 0x0300;
    pot += ADRESL;
    pwm = pot >> 2;

    if(pwm>=0 && pwm<=63 ){
        led_1=1;
    } else{
        led_1=0;
    }
    if(pwm>=63 && pwm<=126){
        led_2=1;
    } else led_2=0;
```

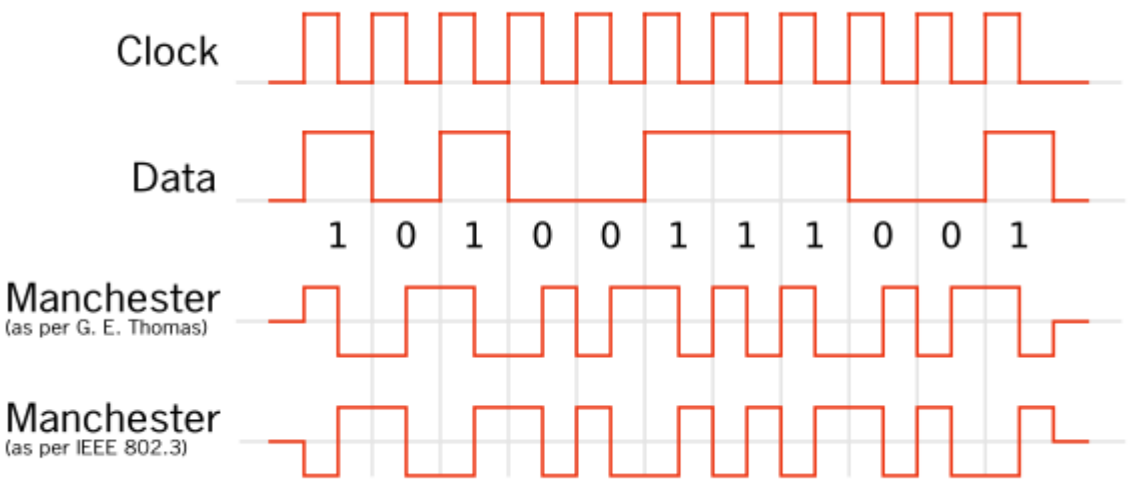
```
 if(pwm>=126 && pwm<=189){
           led_3=1;
        } else led_3=0;


        if(pwm>=189 && pwm<=255){
            led_4=1;
        }else led_4=0;
        __delay_ms(10);
    }
}
```

Nos ahorramos la configuración de bits y de los pines del Pic, con el fin de mostrar lo más importante del código. Lo que se puede ver es un programa que prende un led en una fila de cuatro dependiendo de la posición del potenciómetro. Un programa simple, que implementamos para familiarizarnos con el convertidor. Los dos primeros bits más importantes del registro ADCON0 se encargan de definir el TAD (Tiempo de adquisición de datos) cuyo valor es elegido por recomendación del fabricante dada en la datasheet. Los siguientes cuatro bits, eligen cuál de los 13 canales voy a utilizar, en este caso es el AN0 y el último bit, activa el ADC. Del registro ADCON1 solo importan los primeros dos bits, que definen la forma de guardado del dato analógico que ingresa. En este caso, el conversor es de 10 bits, y está configurado para que guarde los dos bits más significativos en la dirección de memoria ADRESH. Luego con un while condicionado por el bit GO_nDONE, espero a que se complete la conversión, poniéndo él mismo en 0. Guardo los 2 bits de la dirección ADRESH en la variable tipo unsigned int Pot y los corro 8 bits a la derecha, así entran los siguientes 8 bits dado por el conversor almacenados en ADRESL. Por último, en base a este número de 10 bits, realizó una acción (en este caso prender los leds) según su valor actual.

## ¿Cómo se quiere transmitir?

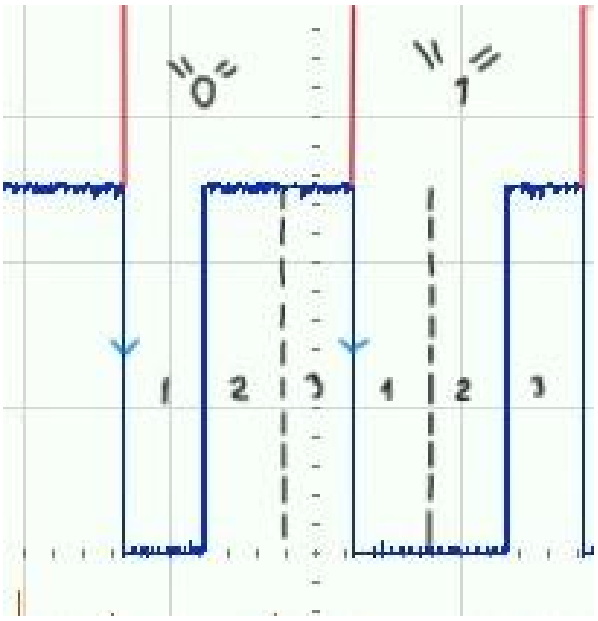
Para empezar, ya se había definido de antemano la utilización del módulo de frecuencia de 433Mhz, universal, barato, y fácil de conseguir. Por ende, cualquier otro criterio, debía ser teniendo en cuenta las limitaciones de dicho transmisor. Usualmente, este tipo de módulos trabajan en baud rates bajos, es decir, de unos 4800 bps, (bits por segundo). Esto, hace que cada bit tenga una duración de 208 microsegundos aprox, y cada trama, es de 10 bits (incluyendo start bit y el stop bit) de 2.08 milisegundos. Esto no es crítico si solo se envía un dato, pero vamos a suponer que se necesita enviar una cadena de 10 bytes. Esto hace un tiempo de transmisión de 20.8 milisegundos. Ahora bien, imaginemos que esos 10 bytes ó 10 datos contienen únicamente 0 's, es decir es una cadena de 10 bytes que contienen 00h. Eso significa que el transmisor está apagado durante 20.8 milisegundos, provocando que el receptor se coloque en un modo inestable generando ruido, y como consecuencia, estaremos recibiendo datos erróneos y no los datos adecuados. Es por esta razón, que se debería optar por un protocolo de codificación el cual garantice un constante cambio de estado, al principio elegimos Manchester, ya que cada bit de información se ven en dos bits codificados. Este método, reconoce como 1 a un flanco ascendente, (0 a 1) y 0, como un flanco descendente, (1 a 0).

Para entender un poco mejor lo dicho previamente, el siguiente gráfico podría ayudar.




Nosotros optamos por el segundo método de Manchester, el cual a diferencia del clásico, toma al 1 como flanco ascendente, y al 0 como flanco descendente. El protocolo del dato, consistirá en una trama que incluya un encabezado de sincronización, un ID de dispositivo, y la información. Posiblemente también incluya un indicador de final de trama. Anteriormente a Manchester, habíamos considerado utilizar la comunicación serie, accesible gracias al periférico UART del pic incluso llegamos a realizar prácticas, pero la comunicación era ineficiente, debido a los problemas mencionados anteriormente.

Por la complejidad que conllevaba codificar en C dicho protocolo, hubo un drástico cambio de planes, y terminamos creando nuestro propio protocolo de envío de datos. Para este protocolo, usamos como referencia al utilizado por los módulos de codificación y decodificación HT12. Este protocolo consta de enviar 3 bits por bit de información, tal que así:




En esta imagen, se observan 2 bits de información, un “0” y un “1”, cada uno, compuesto por 3 bits. La información, o los 3 bits, se empezarán a mandar tras un flanco descendente. Se pueden ver ambos flancos marcados con flechas hacia abajo. Tanto para el “1”, como para el “0”, el primer bit estará en estado bajo, y el último, en alto. El bit que determina la diferencia entre un nivel lógico, y el otro, es el segundo. Si el segundo está en alto, entonces esta mini trama de 3 bits, será un 0, en caso contrario, si el segundo bit está en estado bajo, la mini trama, comunica entonces un 1.

**Codificación y explicación de TX (transmisor).**

Bloque de código 1 (Definiciones, declaraciones de datos y funciones)

```
#define TX PORTAbits.RA7
#define sw_reset PORTAbits.RA4

//POSICIONES DE BIT
#define POS_PL1   0b00000001
#define POS_PL2   0b00000010
#define POS_BT1   0b00000100
#define POS_BT2   0b00001000
#define POS_BT3   0b00010000
#define POS_BT4   0b00100000

// BITS DE SWITCHES
#define PORT_PL1   PORTBbits.RB4
#define PORT_PL2   PORTBbits.RB5
#define PORT_BT1   PORTBbits.RB6
#define PORT_BT2   PORTCbits.RC3
#define PORT_BT3   PORTBbits.RB2
#define PORT_BT4   PORTCbits.RC2

char data_time, i, inter_time, est=0;
int cont;

unsigned int pwm, pot, recibido2, direccion;
unsigned int resultado;

void send_data (void);
void init(void);
void icio (void);


unsigned char info[4];
//info[0]=sync
//info[1]=adelante/atras PWM
//info[2]=izq o derecha
//info[3]=switches
//info[4]=checksum
```

Esta primera parte es sencilla, tan solo se ve la **declaración** de distintas variables y funciones, se **define** un nombre para los pines de los switches e interruptores, y también se les asigna una **posición** de bit para juntar la información de su estado en una variable de 8 bits. La función init, se encarga de inicializar los pines, puertos, registros, etc. La función icio, se encarga de mandar un bit por un tiempo predeterminado, el cual **sincroniza** inicialmente los módulos de comunicación.

Bloque de código 2 (+ Declaraciones, conversiones ADC, usos de funciones)

```
init (); //Configuración de puertos, registros y tal
icio (); //Manda bit de sincronización inicial
unsigned char checksum;  //Dato de 1 byte

info[0]=0b01010101; //Valor usado para sync

checksum=info[1]^info[2];
checksum=checksum^info[3];
info[4]=checksum;

 send_data();
  ADCON0bits.CHS = 0b0010;//AN2
 __delay_ms(5); //Tiempo muerto entre envio de data, y de paso se carga el
 ADC para la próxima conversion
 ADCON0bits.GO_nDONE=1;
  while(ADCON0bits.GO_nDONE==1){}//Espera a que se puedan adquirir los
 datos
 pot = ADRESH << 8; //Corro los 2 bits para que adresL no los pise
 pot&= 0x0300; //Se encarga de que no se llene de basura el dato pot al
 correr 8 bits
 pot += ADRESL; //Se agregan los bits de adresl a pot
 pwm = pot >> 2; //Divido entre 4 para que el maximo sea 255 y no 1024
 partes
 info[1]=pwm;
```

Este segundo bloque de código, muestra el uso de las funciones en orden, la asignación de la información utilizada para sincronizar, también se puede observar la transformación analogica-digital, la cual se hace mediante el periférico ADC del PIC, y necesita un tiempo de reposo para que carguen los capacitores utilizados en el proceso. La otra conversión, para el joystick que maneja la dirección del auto, es prácticamente lo mismo, solo que se cambia el registro `ADCON0bits` para definir el otro canal conectado a dicho potenciómetro. Aquí algo esencial, es la función send_data(), la cual al ejecutarse se encarga del envío de los bits, utilizado datos globales declarados anteriormente, es por eso que no tiene argumento alguno.

Bloque de código 3 (Programación de botones)

```
     //BOTONES
     if(RA4==1)est=0;    //RESET, ES NC
       if(est==0 && RA4==0){
         est=1;
         asm("goto 0"); //Sentencia en asembler, que reinicia el programa al bit
         0 de memoria
     }
     //PALANCA 1
     if (PORT_PL1 == 1){
         info[3] |= POS_PL1; //PONE EN 1 EL BIT DE LA PALANCA
     }
     else{
         info[3] &= ~POS_PL1;//PONE EN 0 EL BIT DE LA PALANCA
     }
     if (PORT_BT1 == 1){
         info[3] |= POS_BT1;
     }
     else{
         info[3] &= ~POS_BT1;//PONE EN 0 EL BIT DE LA PALANCA
     }
```

Este tercer bloque, recortado, muestra cómo se recopila la información del estado de los pulsadores, para luego almacenarla en los bits de la variable `info[3]`. También se puede observar el botón RESET, el cual al ser presionado, resetea desde el byte 0 de memoria al programa entero.

Bloque de código 4 (Función de envío de datos)

```
void send_data (void){/*aca tengo que mandar los bits en tiempo me falta agregar
los bits de protocolo*/
    char a;
    for(a=0; a<5; a++){
        TX=1;
        for (i=0; i<8; i++){
            TX=0;
            if ((info[a]&1)==1){
                __delay_us(508);
                TX=1;
                __delay_us(254);
            }
            else{
         __delay_us(254);
         TX=1;
         __delay_us(508);
           }
            info[a]=info[a]>>1;
        }
    }
}
```

Este último bloque del código del control remoto TX, es la función que se encarga de mandar la trama por el bit TX. Funciona con el protocolo mencionado anteriormente, el cual pone el pin TX en alto por X tiempo para enviar un 1, y por 2X en alto para envíar un 0.

Estos bloques de código mostrados, son solo extractos fundamentales del código principal, el cual será entregado en otro archivo, y no se escribió completamente en el corriente informe debido a su gran extensión.

**Codificación RX (Recepción)**

```
void main (){
  init ();
  while (1){
      if (let==1){//O sea, si pasó 4sync (tiempo) sin recibir ningun bit
          let=0;
          desarme_data();
          uso_de_data ();
          cont=0;
          i=0;
      }
  }
}
```

Aquí se puede ver el void main. Empieza con una condición, `let==1,` la cual solo se cumple si han pasado más de cuatro interrupciones de tiempo, sin recibir ningún bit, por ende se interpreta que ya terminó la trama, y comienza el intervalo de tiempo entre una trama y otra, intervalo el cual se aprovecha para realizar otras funciones. La primera función que se ejecuta, es `desarme_data` la cual tan solo se encarga de asignar los bytes recibidos y almacenados en un arreglo, a una variable con nombre específico, también compara que el checksum sea el mismo. (bloque de código 1) . Luego `uso_de_data,` una de las funciones más importantes, ya que aquí está el pwm y el código que se encarga de mandarle un pulso al servo con ancho variable según el valor recibido del joystick (bloque de código 2).

Bloque de código 1

```
void desarme_data (){
    pwm=get[1];
    direcc=get[2];
    buttom=get[3];
    checksum=get[4];
    see_checksum=direcc^pwm;
    see_checksum=see_checksum^buttom;
    if(see_checksum==checksum){
        ok=1;
        TMR1=0;
        habilit=1;
    }
}
```

Esta sección de código, se encarga de pasar las variables del arreglo, a otra variable con el nombre correspondiente. También se encarga de comprobar el checksum, y en caso de que coincida, habilita el uso de esa información seteando la variable ok en 1. También, se reinicia el timer 1, el cual se encarga de contar el tiempo sin recibir tramas de información.

Bloque de código 2 parte 1

```
void uso_de_data(void){
    unsigned char aux=0;
    if(ok==1 && habilit==1){ //si comprueba el checksum
        ok=0;
        INTE=0; //Interrupciones por rb0
        //MAIN MOTOR PWM
     if(pwm>129 && pwm<256){
           if(pwm>252)pwm=253;
           aux=pwm-129;
           aux=(aux*2);
           P1M1=0; //Pwm hacia adelante
            reverse_light=0;
     }else if(pwm>=0 && pwm<126){
           aux=126-pwm;
           aux=(aux*2)-1;
           P1M1=1; //PWM hacia atras
            if(aux>252)aux=253;
            reverse_light=1; //Luces traseras
     }
```

Primera mitad de este bloque de código 2, mencionado anteriormente, se programó el **PWM enhanced mode**. En dicho modo, se configura por software a través de registros específicos, la posibilidad de generar un pwm en dos pines diferentes, según el valor del bit P1M1, y dos continuas, en otros dos pines diferentes, según el valor del mismo bit. Esto es para poder usarlo fácilmente con un puente H, el cual requiere cuatro entradas. Lo único a tener en cuenta, es que este modo en particular con cuatro pines, no cuenta con **tiempo muerto** entre un pwm y otro, más que el producido por software, es decir, si se pasa muy rápido del 100% de pwm en un sentido, al 100% del otro, podría haber riesgo de cortocircuito, y quemarse así los mosfet.

En sí, las cuentas que se ven, son para asignar los valores a la potencia y sentido de giro, es decir, que del 126-0 tenga un giro **inverso**, con máxima potencia en 0 y mínima en 126, y, que del 129-255, sea al revés, tenga el giro (hacia **adelante**) con mínima potencia en 129 aprox, y máxima en 255, dejando un rango de reposo del pwm entre 126-129, para darle margen fallas del análogo del joystick. También se puede observar que se desactivan las interrupciones del pin RB0, y se activan las luces traseras al ir en reversa.

Bloque de código 2 parte 2

```
if(pwm>=126 && pwm<=129){//Punto medio del joystick
                    aux=0;
        }

        if(aux==0){
            enable=0; //Desactiva el puente H
            reverse_light=0; //Apaga las luces traseras
        }
        else enable=1; //Activa el puente H

        aux/=3;//para limitar la velocidad

        //NITRO
        if(((buttom&1)==1)&&((buttom&32)==32)){ //Cuando el botón que está en la
posición 6 de la variables buttom, y la palanca de la posisción 1, ambos están
en 1, se activa el nitro
            if(P1M1){
                aux=(aux)*3; //Se vuelve al valor no limitado
            }else {
                aux=(aux+1)*3; //Se vuelve al valor no limitado, y se le suma 1
para cuando está en 0
            }
            turbo_light=1;
        }else turbo_light=0;


        //BOTÓN
        if((buttom&2)==2){ //Deben ser las luces de adelante
            front_light=1;  //Se prenden si están en uno
        }else front_light=0;
```

En esta parte del programa, simplemente se muestran las precauciones cuando el joystick está en su punto medio, se muestra la división de la potencia del auto sobre 3, para que esta no esté al máximo, también se muestra que al presionar el botón de nitro, está división se anula, y se ve como se prenden las luces delanteras con una de las palancas del control.

Bloque de código 2 parte 2

```
            if(direcc>=0 && direcc<=36){
                  up=64636; //Pulso 1,8mS

            }

            else if(direcc>36 && direcc<=72){
                  up=64686; //Pulso 1,7mS

            }
            else if(direcc>72 && direcc<=108){
                  up=64736; //Pulso 1.6mS


            }
            else if(direcc>108 && direcc<=120){
                  up=64736; //Pulso 1.6mS

            }
            else if(direcc>120 && direcc<=144){
                  up=64798; //Pulso 1.5mS

            }

          contx++;
          if(contx==10){
             ledtx=~ledtx;
             contx=0;
         }

         T1CONbits.T1CKPS = 01; //Devuelvo preescaler a :2
         marca2=2;
         TMR1IF=0;
         TMR1=up; //Tiempo a reproducir
         servo=1;
         TMR1IE=1; //Interrupciones enable
         TMR1ON = 1; //Habilito Timer
         CCPR1L=aux; //Valor de pwm dado al registro del periferico
    }
}
```

En esta segunda parte, tan solo se le asigna un **ancho de pulso** al servo, en función al valor recibido por el analógico de dirección del PIC de transmisión, hay bastantes más else if, pero no pusimos todos debido a que buscamos explicar lo esencial del código.. Tras esto, se implementa el **TMR1** dedicado completamente a cronometrar el ancho de pulso, para que sea exacto. Se asigna el valor 2 en la variable marca 2, debido a que marca2, define el modo de operación del timer1, que puede cronometrar el tiempo del servo, o el tiempo entre una trama recibida correcta, y otra.También hay un contador que prende el led de confirmación de comunicación, cada 10 tramas correctas recibidas.

Bloque de código 3

A continuación, la parte más compleja del código, su motor, ya qué con las interrupciones se alcanza la precisión para que todo funcione correctamente.

```
void __interrupt () ISR (){//Debo hacer que al detectar un flanco descendente cuente hasta el siguiente flanco ascendente y segun este tiempo es un 1 o 0
    if(T0IF==1 && T0IE==1){ //Para que solo entre cuando está activo el timer
        if(marca1==1){
            if(tiempo==5){ //O sea, solo si el timer no se empezó a restar, tiempo se iguala a 5 cada vez que recibe un pulso, cuando deje de recibir, no se va a poner en 5, y se termino la trama
                get[i]=get[i]>>1; //Corro el bit hacia la derecha
                if(RX==0){
                    get[i]+=128;
                }
                cont++; //Suma de flancos q se van guardando
                if(cont==8 && i<cant_tramas){ //Para que cada 8 bits, se cambie el índice del array, pero que no lo cambie al realizar los ultimos 8 bits, por eso menor a 4
                    i++;
                    cont=0;
                }
            }
            if(tiempo>0){
                tiempo--;
                if(tiempo==0){
                    let=1;
                }
            }
```

Con las interrupciones, lo que se logra, es detectar los momentos exactos de los flancos de la información, medirlos temporalmente, y a partir de esa medición precisa de tiempo, determinar si el bit es un 1 o un 0. Al llegar a los 8 bits, se cambia el índice del array, así, hasta llegar a los 5 bytes, que es el total de información que se envía desde el control remoto RF. Este proceso está controlado, es decir, se controla el tiempo entre un bit y otro, si este tiempo es mayor al que debería según lo programado en el control RF, significa que se terminó el envío de la trama de datos. Si terminó, se pone en 1 la variable LET, permitiendo seguir con el proceso de desarme de datos.

Bloque de código 3 parte 2

```
         }else{
            if(sync>0){
                sync--;
                if(sync==0){ //cUANDO LLEGUE A 0, ACTIVA EL MODO DE LECTURA DE BYTE, PORQUE TECNICAMENTE YA SE SINCRONIZÓ
                    INTEDG=0; //RB0 activado por flanco descendente
                    marca1=1;
                    T0IE=0;
                }
            }
        }

        T0IF=0; //bajo bandera timer
    }
```

Aquí se programó la sincronización, es un timer iniciado en la primera ejecución del programa, tras detectar el último bit de la trama de datos, al cumplirse un x tiempo, significa que ya está por venir la siguiente trama, y es ahí donde empieza a funcionar el programa entero.

Bloque de código 3 parte 3

```
if(INTF==1 && INTE==1){
   if(marca1==0){
            if(T0IE==0){
                T0IE=1; //Activa el timer
            }
            CCPR1L=0;
            sync=time_sync; //este valor será 30 hasta q deje de detectar flancos, empiece a contar, y al tiempo sin flancos, se sincroniza
        }
        else{
            T0IF=0;
            T0IE=1;
            TMR0=76; //180*2uS=360uS
            tiempo=5;//Valor que regula el tiempo entre bits
        }
        INTF=0; //Pone en 0 bandera de interrupciones
    }
```

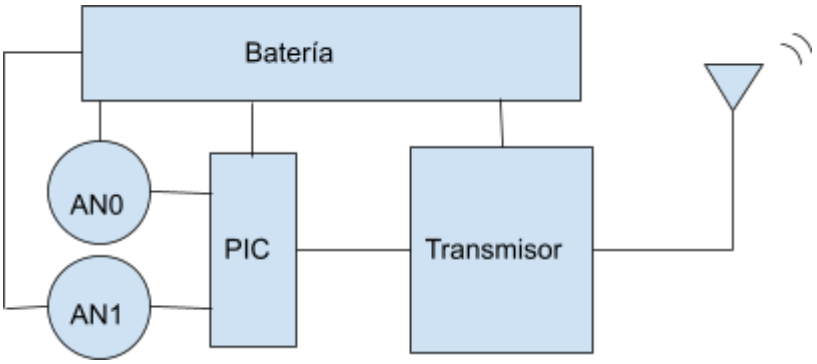
Acá se puede ver la activación del timer al detectar el flanco descendente, el cual se usa para medir si es un 1 o 0. Por otro lado, en el ELSE, se declara el tiempo que mide si se dejaron de detectar bites.

Bloque de código 3 parte 4

```
 if(TMR1IE==1 && TMR1IF==1){
        TMR1IF=0;
        if(marca2==1){ //Si entra con marca2==1 es porque se cumplió el tiempo

            if(CCPR1L>0){
                CCPR1L=0;
            }
            get[1]=0; //Mando todo a 0 por las dudas
            get[2]=128;//Mando todo a 0 por las dudas
            get[4]=0;//Mando todo a 0 por las dudas
            ok=0;//Mando todo a 0 por las dudas
            habilit=0;//Mando todo a 0 por las dudas
            enable=0;
            asm("goto 0"); //Que resetee todo el pic si se cumplio el timer 1
que son 524mS con el preescaler :8 sin recibir ninguna trama
        }
        if(marca2==2){
            servo=0;
            T1CONbits.T1CKPS = 0b11; //Preescaler en 8 para iniciar timer de
desincronización
            TMR1=20000; //360ms
            marca2=1; //Que ponga en modo timer sincronización
            INTE=1;//Prueba
            T0IE=1;//Prueba
        }
    }
}
```

En esta última parte, se configura el timer 1, utilizado para medir el tiempo sin recibir una trama, es decir, si se desincroniza con el control, por ejemplo. Si este tiempo es mayor a 360mS, se reiniciará el programa, y deberá dejar inmovil al auto. Por otro lado, se usa también para mandar el pulso del servomotor, que es tras este pulso que se inicia el timer de 360mS.

## Diagrama de bloques




### Batería

Para este apartado decidimos utilizar baterías de Li-Po recargables, ya que nos dan una gran versatilidad debido a que cuentan con una gran capacidad. Utilizamos dos en serie de 3,7V cada una, con una capacidad de 650mAh, quedando así una batería de 2 celdas a 650mAh.


### Transmisor


En el caso de la transmisión de la información, utilizamos un módulo RF de 433Mhz. El cálculo de la antena a utilizar es el siguiente:

$$\lambda = \frac{V}{f} = \frac{300000\ km/s}{433MHz} = 0,69m = 69cm$$

$\frac{\lambda}{2} = 34,5cm$ longitud de la antena

### Analógicos

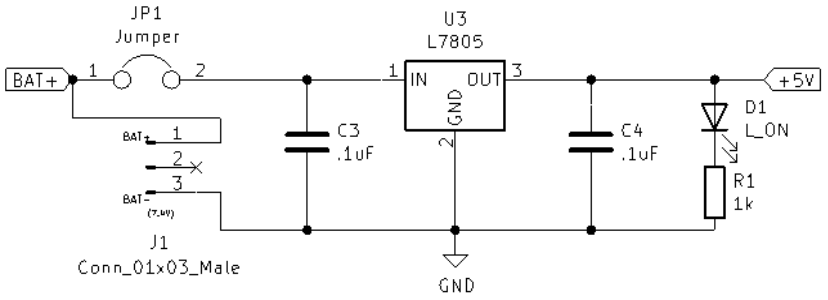
Utilizaremos dos joysticks analógicos, uno se encargará de la dirección del auto mientras que el otro será utilizado para la velocidad y el sentido de giro del motor.

### Circuito Esquemático y PCB

Para la creación del circuito esquemático y el diseño de la PCB utilizamos KiCad, debido a su versatilidad y accesibilidad. Optamos por este software de código abierto ya que estaba disponible en las computadoras de la escuela y además, contábamos con experiencia previa en su uso durante los últimos años de la carrera. Esta familiaridad con la herramienta nos permitió agilizar el proceso de diseño y garantizar una mayor eficiencia en la creación de los esquemáticos y la disposición de los componentes en la placa de circuito impreso.

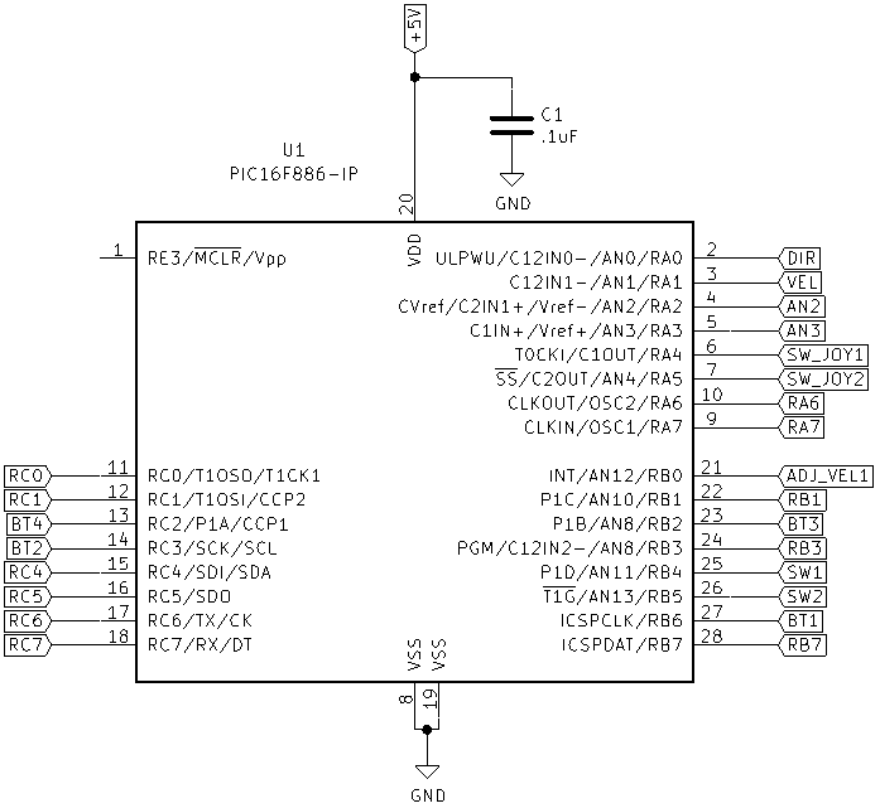
## REGULADOR +5V

Para adaptar la tensión nominal de la batería de 7,4V y alimentar al microcontrolador y otros componentes, empleamos un regulador lineal LM7805. Esta elección se basó en el bajo consumo del circuito en su conjunto, lo que aseguraba una baja disipación de potencia en el integrado. Además incorporamos capacitores de filtrado y un LED indicador de encendido para brindar retroalimentación visual sobre el funcionamiento del joystick.




## MICROCONTROLADOR

Para filtrar el posible ruido en la alimentación del microcontrolador PIC16F886, añadimos un capacitor de tantalio de 100nF. Además, decidimos soldar cada uno de los pines del microcontrolador a diversos componentes. Lo que nos brinda la posibilidad de expansión en el futuro.

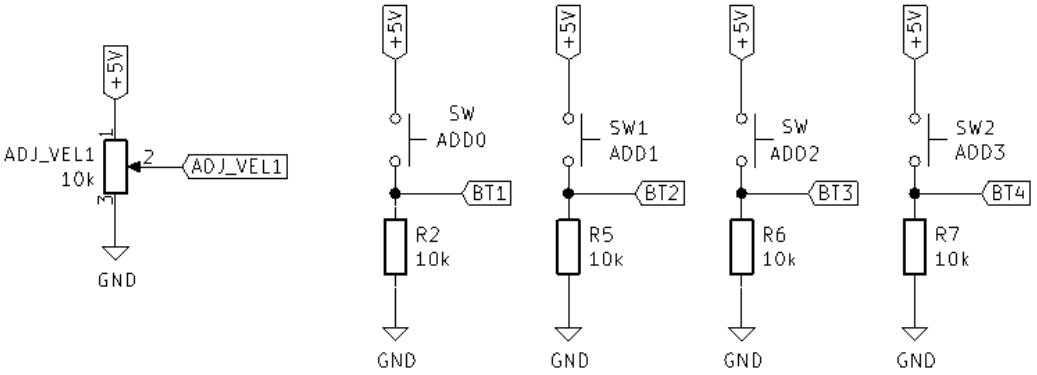


## PULSADORES

Optamos por instalar 4 pulsadores en una configuración PullDown. Esta disposición asegura que, cuando los pulsadores no están presionados, el microcontrolador reciba un nivel lógico bajo, representado por un "0" lógico. Al presionar un pulsador, el microcontrolador recibe un nivel lógico alto, representado por un "1", lo que indica la activación de la función correspondiente.

De los 4 pulsadores, asignamos dos para acciones específicas: uno se utilizará para activar una función de nitro incorporada al vehículo (explicación en el código), mientras que el otro será responsable de encender el display indicador del nivel de batería colocado en el auto.

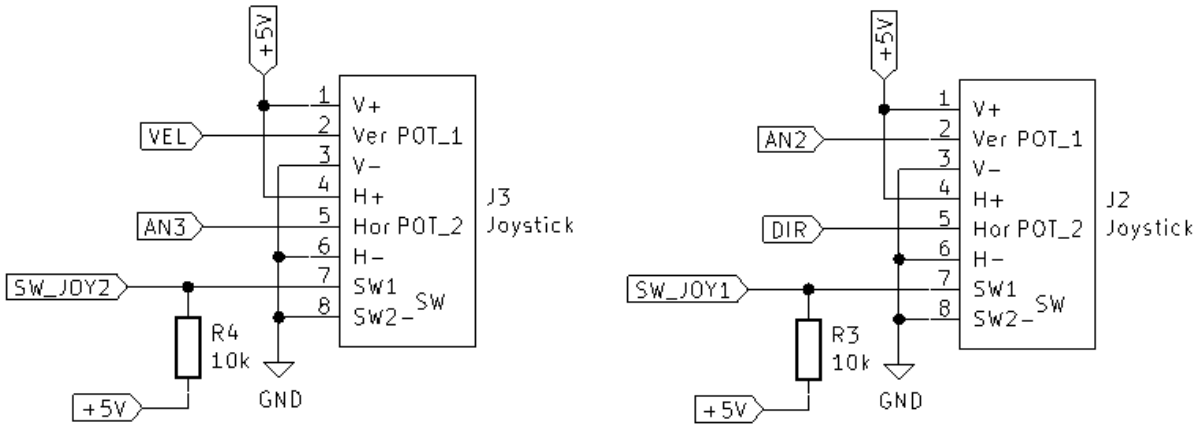
Además, hemos incorporado un potenciómetro denominado ADJ_VEL1 con la idea de emplearlo en el futuro para controlar, por ejemplo, un servomotor que manipule una cámara.




## JOYSTICKS

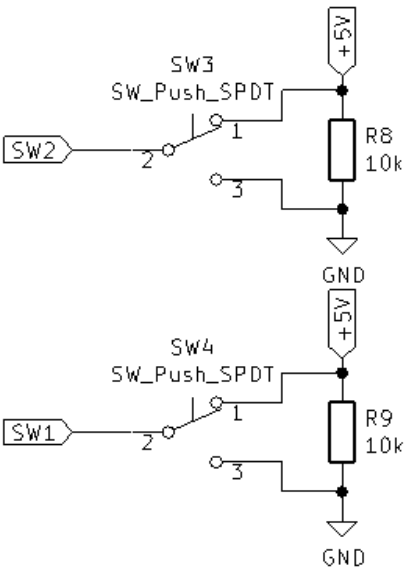
Decidimos emplear dos joysticks para controlar tanto la dirección de giro del vehículo como su velocidad. Cada joystick contiene dos potenciómetros, lo que nos brinda un total de 4 potenciómetros disponibles. En nuestro diseño actual, empleamos dos de estos potenciómetros para nuestros fines actuales, dejando los dos restantes conectados al microcontrolador para posibles usos futuros.

Además, implementamos resistencias de PullUp en los pulsadores incorporados en cada joystick. Uno de estos pulsadores se asignó específicamente para la función de resincronización del control, permitiendo restablecer la conexión o sincronización entre el control y el vehículo en caso de ser necesario.



## INTERRUPTORES

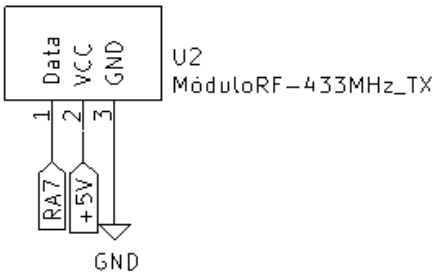
Instalamos dos interruptores de palanca con el propósito de controlar diferentes funciones del vehículo de forma clara y sencilla. En una de las posiciones de estos interruptores, el microcontrolador recibe un nivel lógico bajo, representado por un "0" lógico, mientras que en la otra posición recibe un nivel lógico alto.




Asignamos específicamente uno de estos interruptores para activar las luces delanteras del vehículo. El segundo interruptor de palanca lo utilizamos como una medida de seguridad para habilitar la función de nitro. Esta elección proporciona un mecanismo de control adicional para asegurar que la función de nitro se active únicamente cuando se requiera.

## MÓDULO TRANSMISOR

El módulo transmisor únicamente requiere de una alimentación de +5V. Respecto a la antena, calculamos su longitud con base en la mitad de la longitud de onda de la frecuencia de trabajo del módulo. En este caso, determinamos que una antena de 34 cm de largo era la más adecuada para nuestro propósito. Optamos por utilizar una antena de una antigua radio que encontramos, la cual se ajustaba a la longitud requerida.

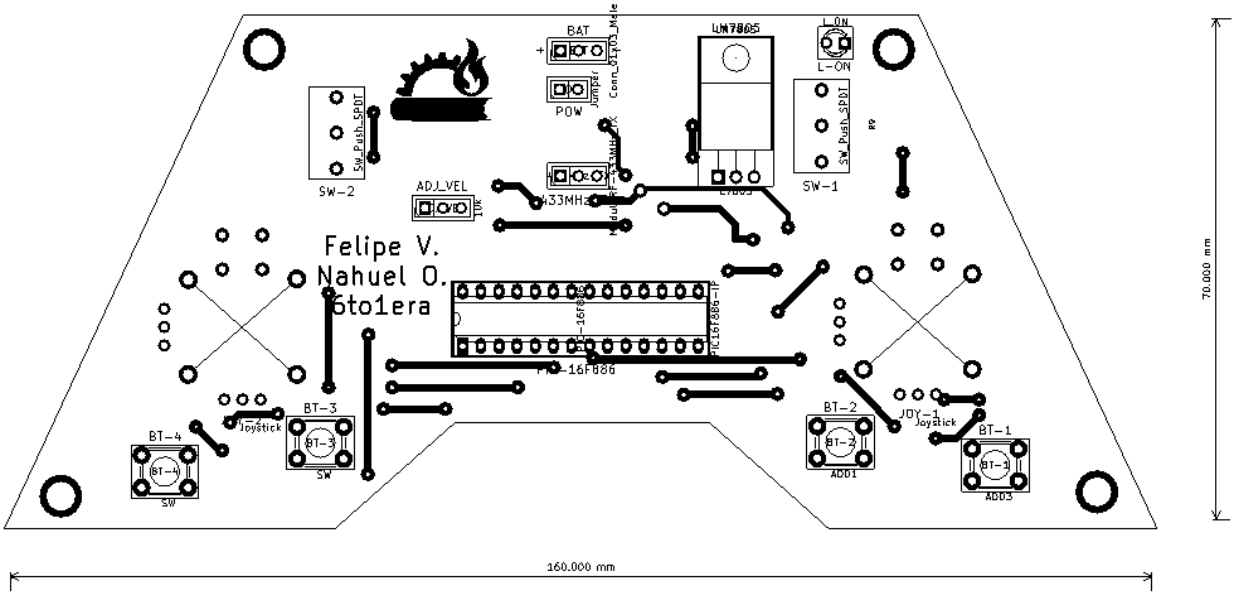


## PCB

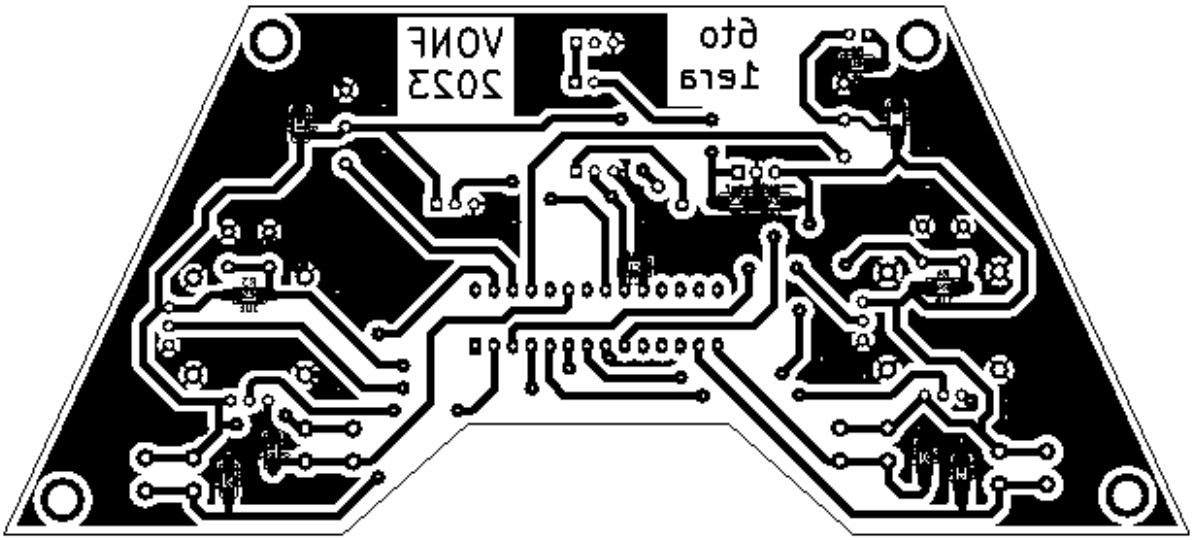
A continuación, en las diferentes vistas de la PCB, se puede apreciar que hemos adoptado por una combinación de componentes SMD (Surface Mount Device) y componentes THT (Through-Hole Technology) con el objetivo de lograr una placa de circuito impreso lo más compacta posible.

Debido a la complejidad y a la alta cantidad de conexiones requeridas para el funcionamiento del circuito, hemos empleado puentes por el lado de los componentes para interconectar los diversos elementos.

### Vista frontal

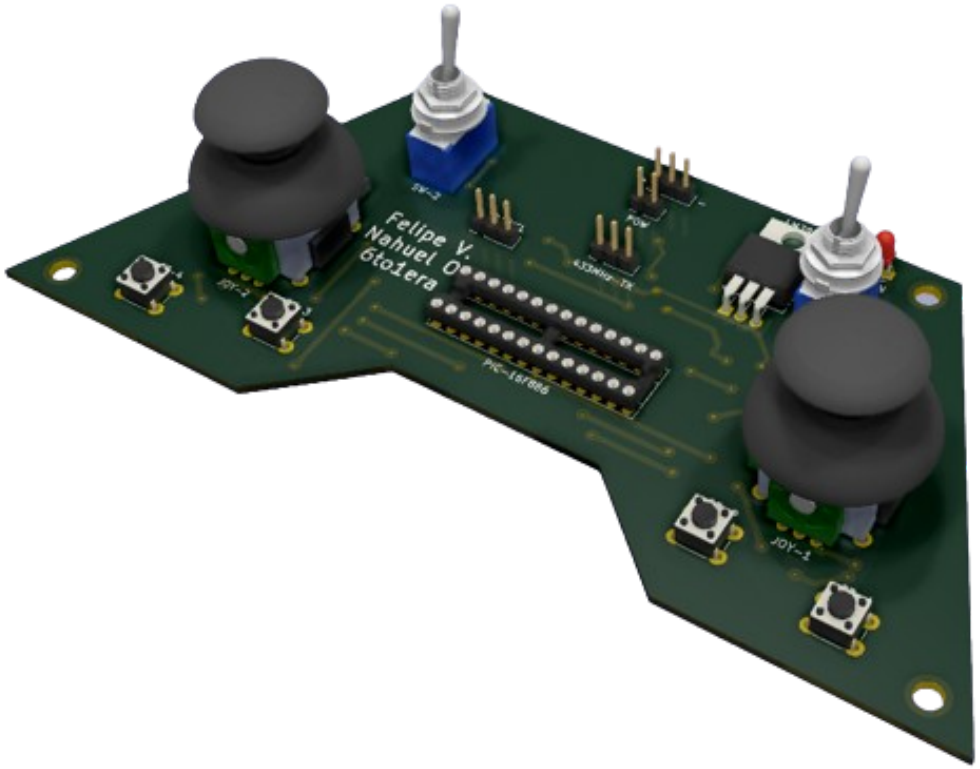



### Vista trasera

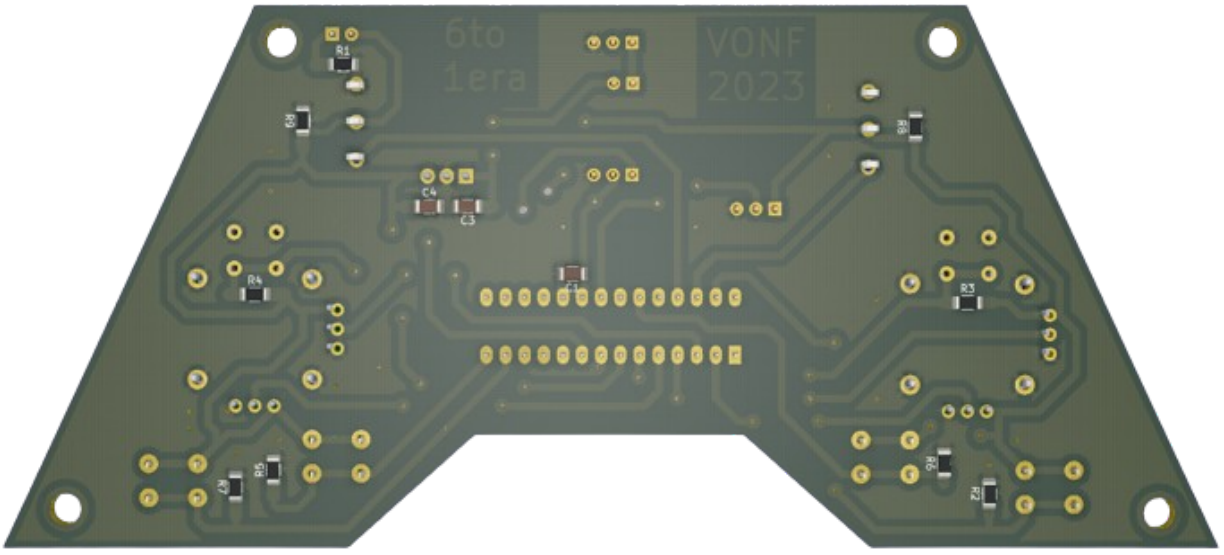


## Renders

**Frontal:**

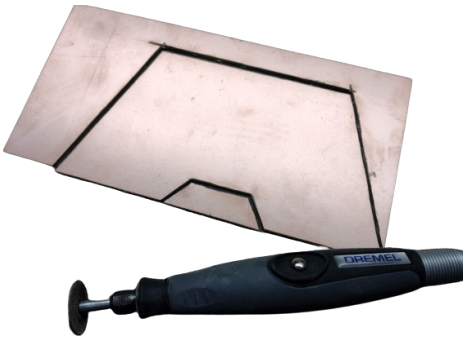



**Trasero:**



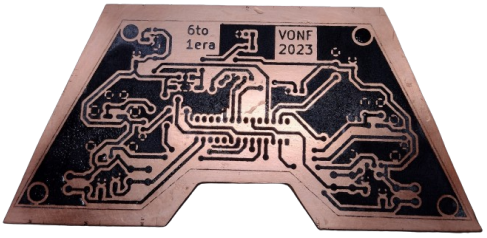
## Proceso de fabricación de la PCB

**Corte de la placa virgen de cobre:** Utilizamos un minitorno para agilizar el proceso de corte de la placa de cobre. Una vez cortada, se procedió a pulir la superficie con virulana para obtener una superficie lisa y preparada para el siguiente paso.




**Transferencia del diseño de pistas a la placa:** Colocamos sobre la placa pulida un papel con la impresión del diseño de pistas realizado previamente en KiCad. Utilizamos una plancha para transferir por medio del calor la tinta desde el papel hacia el cobre de la placa. Posteriormente, sumergimos la placa en agua para facilitar la remoción de los restos de papel, dejando las pistas de tinta sobre el cobre.

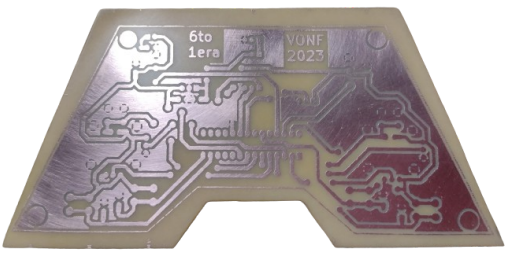
Eliminando papel




Transferencia exitosa de la tinta

**Ataque químico con ácido férrico:** Una vez transferida la tinta se sumerge la placa en ácido férrico, el cual corroerá las partes no cubiertas por la misma. Quedando así, las pistas de cobre según el diseño previamente impreso.

**Limpieza y protección:** Una vez retirada la placa del ácido, se enjuaga cuidadosamente para eliminar cualquier residuo del ácido. Luego, se retira la tinta sobrante para dejar al descubierto las pistas de cobre. Para finalizar, se aplica una fina capa de colofonia (resina de pino) disuelta en alcohol isopropílico para proteger la placa contra el óxido y facilitar el proceso de soldadura de los componentes.



**Grabado de detalles adicionales:** Con el fin de mejorar la estética y la identificación, grabamos detalles como nuestros nombres y el logo del colegio en el lado de los componentes de la placa. Para grabar estos detalles en la superficie de la placa, realizamos un proceso similar a los pasos anteriores, excluyendo el uso del ácido.

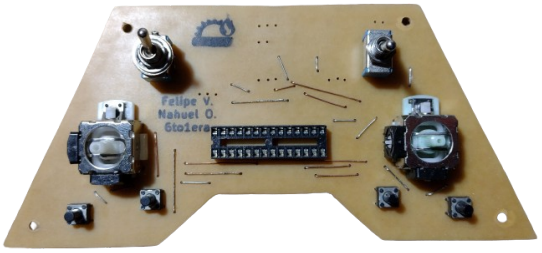

**Comprobación de pistas:** Finalmente, utilizando un multímetro en su función de continuidad, comprobamos cuidadosamente cada pista, asegurándonos de que todas estuvieran correctamente conectadas. Para evitar posibles daños a los componentes debido a errores en la transferencia de la tinta durante la etapa de fabricación..

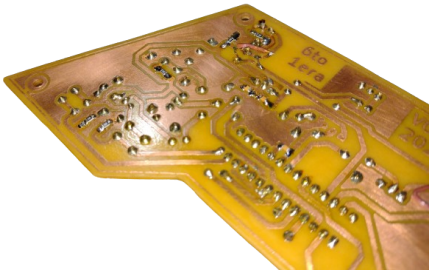
## Proceso de soldado de componentes

El proceso de montaje de los componentes en la PCB se llevó a cabo con cuidado y precisión. Utilizando un minitorno, perforamos la placa con distintas medidas de mechas para adaptarse a las necesidades de cada componente. Empleamos mechas de 0,5mm para los puentes de alambre telefónico, 0,75mm para las patas del microcontrolador, LED y pulsadores, y de 1mm para los joysticks, regulador e interruptores.


Inicialmente, dedicamos tiempo a cortar y preparar, para luego soldar los puentes de alambre telefónico. Posteriormente, procedimos a colocar los joysticks, los cuales requirieron una atención especial. Diseñamos los footprints de estos componentes debido a la falta de modelos estándar disponibles en internet. Al realizar manualmente las perforaciones, nos enfrentamos a cierta dificultad para alinear todos los pines de los joysticks, pero finalmente logramos colocarlos en su sitio.

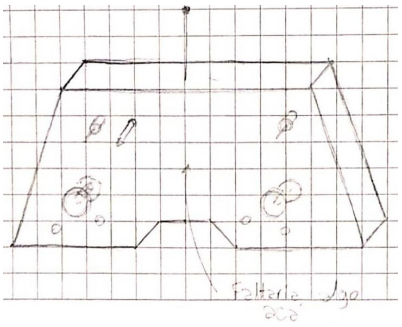
Una vez soldados todos los puentes y componentes THT, nos enfocamos en los componentes SMD. Este proceso, aunque más delicado, se llevó a cabo con cuidado y precisión para garantizar un correcto montaje.



**Diseño de la carcasa**

El diseño de la carcasa para nuestro proyecto fue desarrollado desde cero, teniendo en cuenta las dimensiones y la disposición de la PCB para asegurar un ajuste perfecto. Nuestro objetivo principal era crear una carcasa única y ergonómica que proporcione comodidad al usuario durante su manipulación.



Para la creación del diseño 3D, utilizamos el software Fusion 360, tomando en consideración aspectos como la estética, la funcionalidad y la comodidad.

El material elegido para imprimir la carcasa fue el PLA, conocido por su durabilidad y facilidad de impresión en 3D. Después de múltiples revisiones, modificaciones y mejoras, alcanzamos un diseño final que cumplía con nuestros objetivos de diseño ergonómico y estético.

**Renders**



Nuestra carcasa diseñada meticulosamente incorpora características prácticas y funcionales para mejorar la experiencia del usuario. Entre las características destacadas se incluyen etiquetas identificadoras para cada uno de los botones utilizados, lo que facilita la comprensión de las diversas funciones del dispositivo.

Además, hemos integrado un corte frontal que revela el microcontrolador utilizado, permitiendo una visión directa del corazón del dispositivo. Esta característica no solo agrega un aspecto visual interesante, sino que también sirve como punto de referencia para los usuarios interesados en la tecnología utilizada en el proyecto.

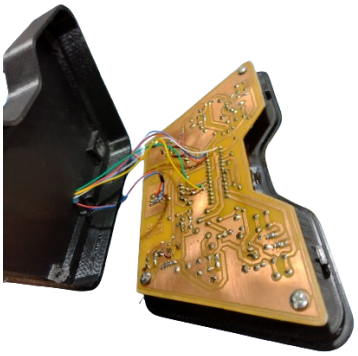
En la parte trasera de la carcasa, hemos implementado un compartimento con una tapa removible que brinda acceso directo a la batería. Este diseño práctico facilita el proceso de carga de la batería, permitiendo un acceso rápido y sencillo sin necesidad de desarmar toda la carcasa.

Además, hemos incorporado un conector hembra que permite la conexión directa de un programador PICKIT. Lo que lo hace especialmente útil, ya que posibilita la actualización directa del programa en el microcontrolador sin la necesidad de extraerlo del zócalo, simplificando y agilizando el proceso de programación.

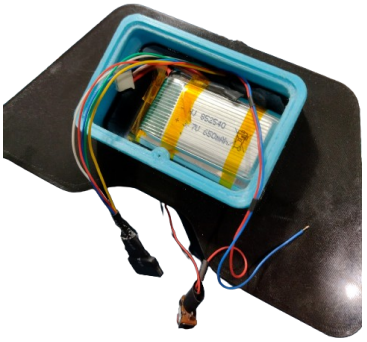
**Diseño real**

Cómo el enfoque para diseñar el joystick se centró en optimizar su facilidad de ensamblaje y desmontaje, ideamos trabas específicas para asegurar que la tapa se acople de manera segura a la parte trasera del joystick. La PCB se posiciona estratégicamente en la parte frontal del mismo y se fija con tornillos M3.

Esta configuración nos permite tener un acceso rápido y sencillo a la PCB cuando sea necesario, simplificando el proceso de mantenimiento, modificación o reparación del joystick.



Colocación de PCB



Compartimento de batería


Vista frontal


Vista Trasera

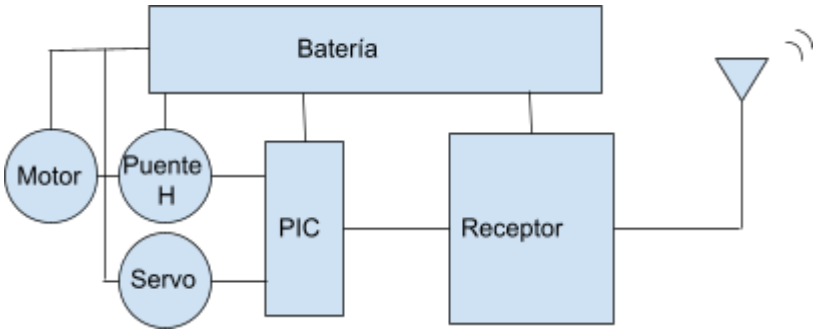
## Electrónica del auto

La electrónica del auto no es nada compleja. El cerebro del mismo se basa en el pic16F886, el cual tendrá dentro el programa con el protocolo para la recepción de datos tipo HT12, el control del servo, y del motor de continua mediante un driver puente H.

En el caso del motor, tuvimos que decidir entre dos tipos distintos, los motores Brushed (con escobillas) ó Brushless (sin escobillas). Terminamos optando por utilizar un motor con escobillas, ya que si bien por tenerlas, son menos eficientes, tienen menor velocidad y requieren de un mantenimiento si se utiliza prolongadamente, a diferencia de los otros, resultan más sencillos de controlar, tanto la velocidad como su sentido de giro. Además son más económicos , poseen un mayor torque y son más controlables a baja velocidad.

El control del servo en principio lo íbamos a realizar utilizando el periférico PWM del propio pic, pero debido a que el servo se controla con una señal pwm de 50Hz, y el mínimo del periférico es de 200Hz aproximadamente, son incompatibles, es por eso que optamos por hacer nosotros mismos el pwm con el timer 0.

### Diagrama de bloques



**Batería**

En este apartado teníamos varias opciones para elegir, pero entre ellas finalmente decidimos utilizar una Batería de Li-Po de 7.4V con una capacidad de 2200mAh y una tasa de descarga de 50C.


La tasa de descarga "C" es un parámetro que indica cuán rápido se puede descargar la batería de manera segura sin que la misma se dañe. Conociendo este parámetro más la capacidad en Amperes, podemos calcular la corriente máxima de descarga. En nuestro caso multiplicamos 2,2A x 50C = 110A. Esto quiere decir que nuestra batería puede entregar una corriente de 110A de manera continua de forma segura.

**Motor**

En el caso del motor, nos decidimos por utilizar uno con escobillas y 35T. Este parámetro indica la velocidad y la fuerza del mismo. A menor T, el motor tendrá más velocidad pero por contra menor fuerza o torque. Caso contrario, presentará una menor velocidad y mayor fuerza. En nuestro caso elegimos uno de parámetro intermedio.


**540 MOTOR 35T PARAMETER**

| | NO LOAD | Rotor block |
|---|---|---|
| Current (A) | ≤1.5A | ≥50A |
| RPM | 12500±10% | |
| Test Voltage (V) | 7.2V | |

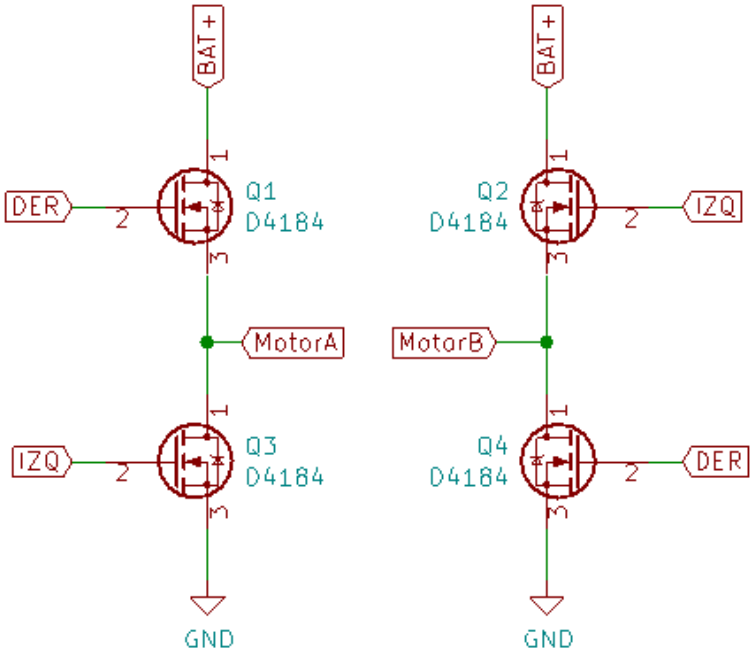
Cómo se ve en la imagen superior, que nos brinda el fabricante, la corriente con carga es mayor a 1,5A, lo que nos limita a utilizar el driver puente H L298N. Debido a esto, elegimos diseñar y utilizar un puente H con MOSFET, en el siguiente apartado se explicará el diseño del mismo.

**Puente H**

Para su diseño, desarrollado completamente por nosotros mismos, se nos presentaron muchas dificultades, principalmente por la falta de disponibilidad de componentes. Finalmente conseguimos unos módulos que traen MOSFET de canal N D4184 Los mismo tienen una RDS <7mΩ, lo que provoca una muy baja caída de tensión, en consecuencia menor sobrecalentamiento. Por otro lado, su corriente ID de manera continua es de 40A.

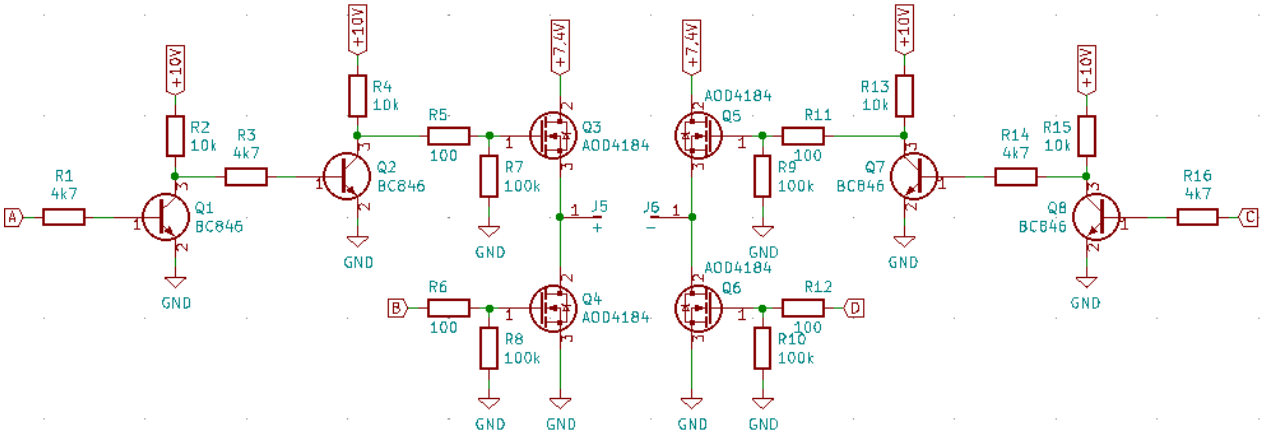
Su tensión VGS mínima a aplicar para que entre en estado de conducción es de 2,6V, por lo que se puede controlar directamente desde el microcontrolador.

Esquema simplificado del puente H:




En este esquema simplificado de un puente H, se puede observar que para que el motor gire hacia la Derecha deberá entrar en conducción Q1 y Q4, y en sentido contrario, Q2 y Q3. Para la variación de la velocidad, se enviará una señal con un ancho de pulso variable.

Nuestro primer circuito realizado y ensayado fue el siguiente:



**Explicación del funcionamiento del circuito:**

Las señales B y D, son las señales de PWM que controlan la velocidad de giro del motor.

Por otro lado, las señales A y C, son niveles de continua (0 ó 1 lógico) que hacen conducir o no a cada rama determinada. Esto se consiguió gracias al periférico PWM del PIC, específicamente su función de PWM enhanced mode full bridge, dicho modo genera dos señales continuas, cada una producida a la par de la otra señal con modulación en el ancho de pulso

Para un determinado sentido de giro del motor debe entrar en conducción Q3 y Q6, y para el otro sentido de giro Q5 y Q4.

R7, R8, R9 y R10: Estas resistencias son utilizadas para establecer un nivel lógico de 0V en la puerta de cada MOSFET (Q3, Q4, Q5 y Q6), quedando cada uno en corte. Esto es necesario ya que cuando se enciende el circuito el microcontrolador puede enviar un pico de tensión positiva en alguno de sus pines, lo que conlleva a una saturación de alguno de los transistores.

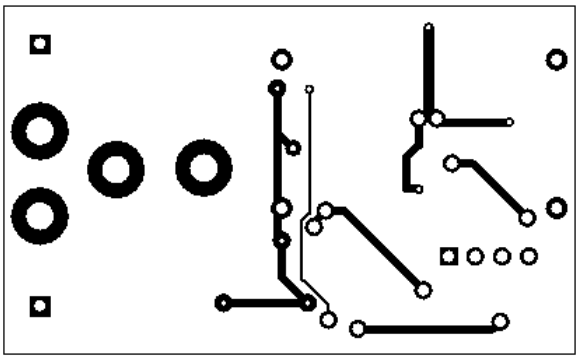
R5, R6, R11 y R12: Estas resistencias son únicamente limitadoras de corriente.

Q1, Q2, Q7 Y Q8: Estos transistores son los encargados de adaptar el nivel de continua entregado por el microcontrolador para que los MOSFETs puedan ser saturados. La tensión en el gate de Q3 y Q5 respecto a masa, según el fabricante debe ser de +2,4V. Por lo tanto es necesario, mediante esta etapa con transistores, elevar la tensión que entrega la salida del microcontrolador, a un nivel mayor a 7,4V.

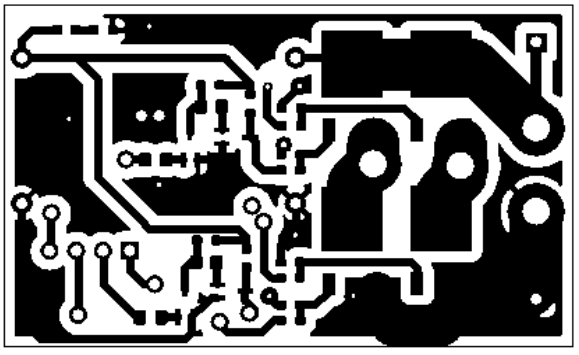
Estos +10V fueron tomados desde una fuente StepUp conectada a la batería de 7,4V.

Una vez ensayado el circuito en protoboard, decidimos realizar una pcb, ya que el mismo funcionaba correctamente. La PCB utilizaba en su totalidad componentes de tipo SMD.

**PCB:**

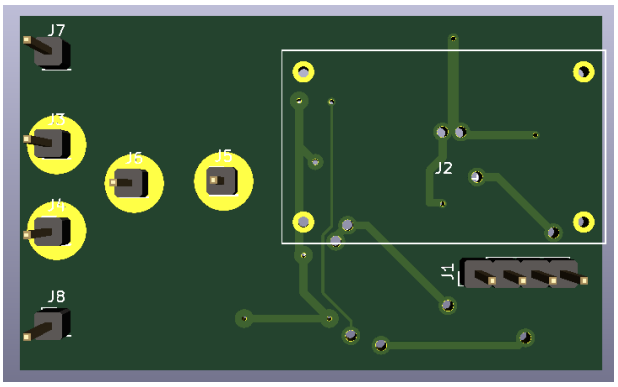


Vista Frontal

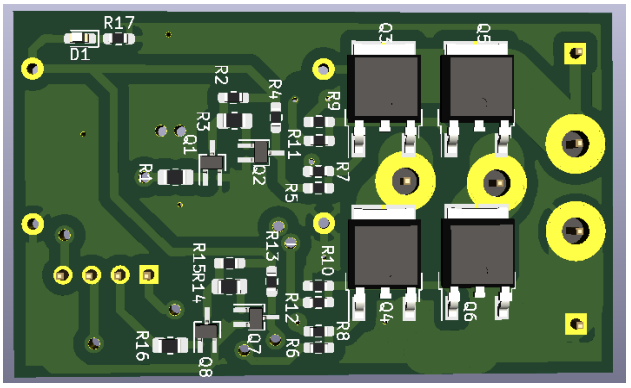


Vista Trasera

**Vistas 3D:**

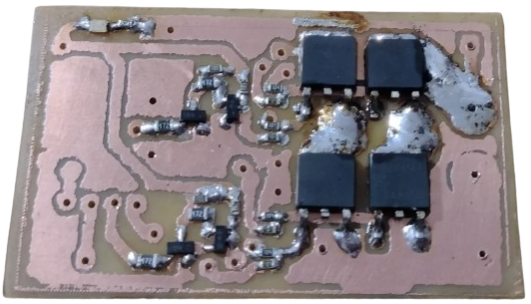



Vista Frontal

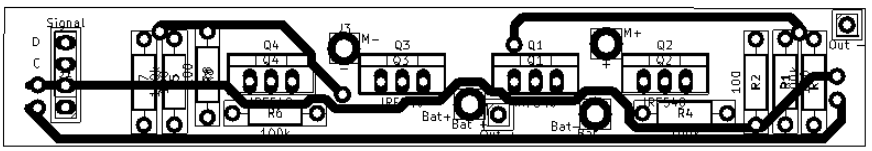



Vista Trasera

Una vez soldados todos los componentes, decidimos conectar el motor, pero al conectar las señales de entrada a cada uno de los MOSFET los mismos se nos quemaron. Esto sucedió debido a un error al conectar las señales: A, B, C y D. Lo que generó que entren en conducción Q3 y Q4 en simultáneo, provocando un cortocircuito entre +VCC y GND de la batería, como la batería es capaz de entregar gran cantidad de corriente, la misma superó los límites del MOSFET, provocando su rotura.

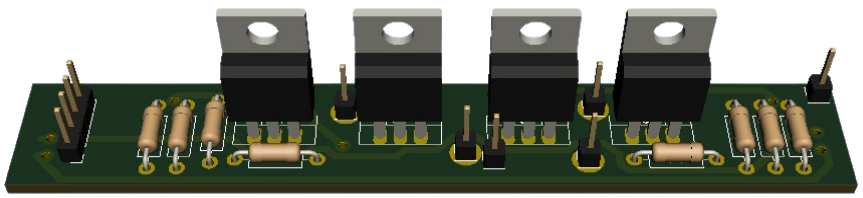


Debido al alto costo y la escasez de los MOSFETs en el país, optamos por comprarlos en China. Mientras esperábamos su entrega, decidimos diseñar una nueva placa basada en los MOSFETs IRF540, los cuales estaban disponibles localmente a un precio más razonable. Una vez adquiridos, procedimos a diseñar y fabricar una nueva PCB, y luego los soldamos a la placa.




Vista frontal

Vista trasera



Vista 3D

Lamentablemente, al conectar el motor al circuito, observamos un aumento significativo en la temperatura de los MOSFETs debido al alto consumo de energía del motor. Esta situación se volvió inviable para nuestro proyecto, ya que la temperatura alcanzaba niveles preocupantes en un corto período de tiempo. A pesar de considerar el uso de un disipador, su tamaño requerido resultó ser demasiado grande, por lo que descartamos la instalación en el chasis del auto RC.
Con el tiempo agotándose y viendo que los MOSFETs pedidos en China llegarían muy sobre la fecha de exposición, tomamos la decisión de adquirir un módulo de puente H de 46A como una solución alternativa.

**Conexiones del módulo**


+5V y GND: Destinados a la alimentación del circuito lógico.

RPWM y LPWM: Estos pines están diseñados para recibir señales PWM del microcontrolador. La señal PWM enviada a estos pines controla la velocidad y dirección del motor.

R_EN y L_EN: Utilizados para habilitar o deshabilitar el funcionamiento del módulo. Para activar el movimiento del motor en una dirección específica, es necesario enviar una señal lógica "1" a estos pines.

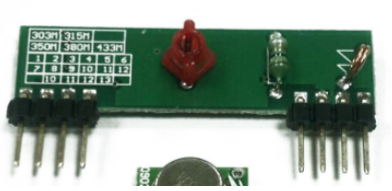
R_IS y L_IS: Aunque no se utilizan para controlar el motor directamente, estos pines pueden emplearse para medir la corriente del motor, brindando información valiosa sobre el consumo y rendimiento del mismo.

Además de estos pines de conexión, el módulo dispone de una bornera donde se conecta la alimentación proveniente de la batería, así como los cables de conexión del motor. Por ejemplo para lograr que el motor gire en dirección derecha, se debe enviar una señal de PWM al pin RPWM y colocar los pines R_EN y L_EN en estado lógico "1".

**Servo**


Para el caso del servomotor, el cual controlará la dirección de las ruedas delanteras, decidimos utilizar el MG996R, ya que brinda un alto torque además de contar en su interior con engranajes metálicos, lo que se traduce en una mayor durabilidad. Su tensión de alimentación es de entre 6,8V-7,2V y su consumo de corriente llega hasta picos de 1,5A por lo tanto, por este elevado consumo utilizamos una fuente StepDown (fuente switching), regulada en 7V para reducir el nivel de tensión de la batería.

**Receptor**

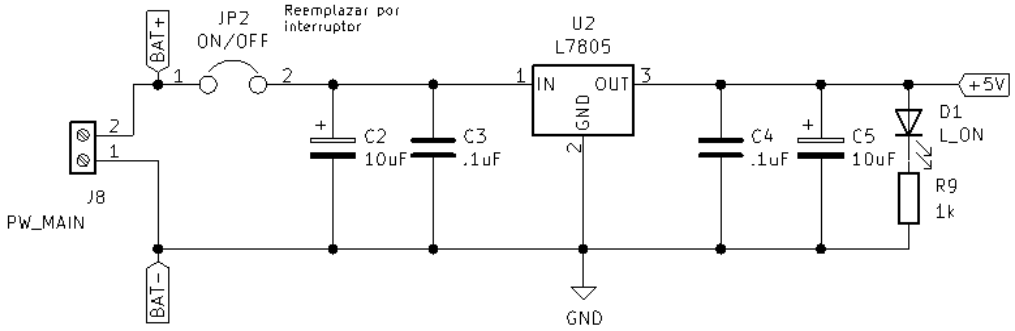


Para el receptor utilizamos un módulo RF de 433Mhz. Para el mismo utilizaremos una antena de igual longitud a la del transmisor.

Este receptor se encarga de recibir por protocolo OOK (ON OFF KEYING) es decir, recibe una señal análogica, y la codifica como un 1 lógico, y al no recibir, se considera un 0.

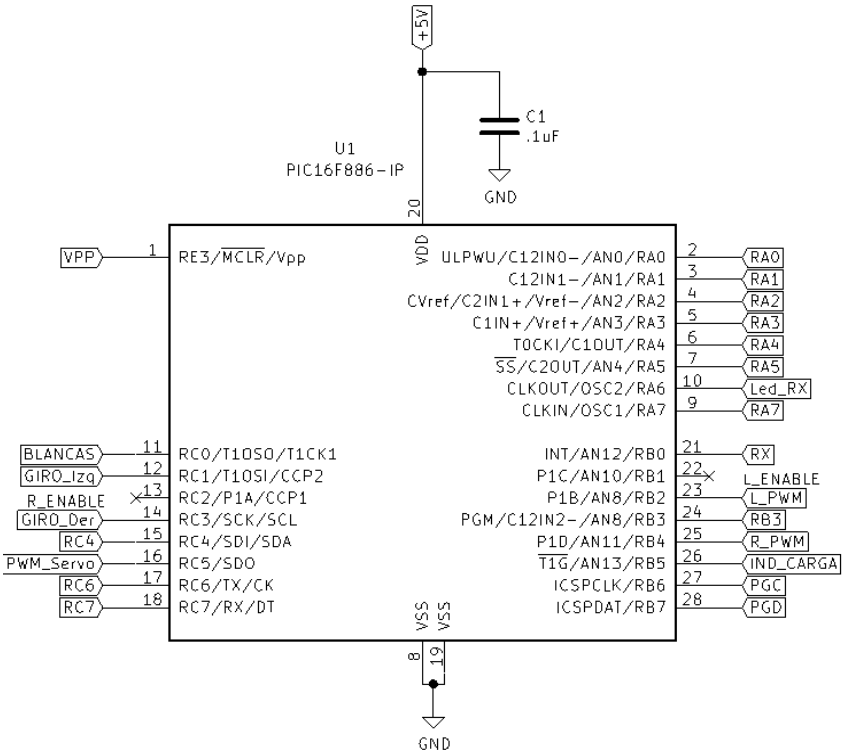
## REGULADOR +5V

Para alimentar la parte lógica de control de la placa del auto RC, decidimos utilizar una fuente con un LM7805 al igual que en el joystick, además de los capacitores de tantalio, le agregamos capacitores electrolíticos de mayor capacidad, ya que esta placa presentará un mayor consumo en comparación a la del mando. Además incorporamos un led indicador de encendido y un interruptor para desenergizar toda la parte de control.



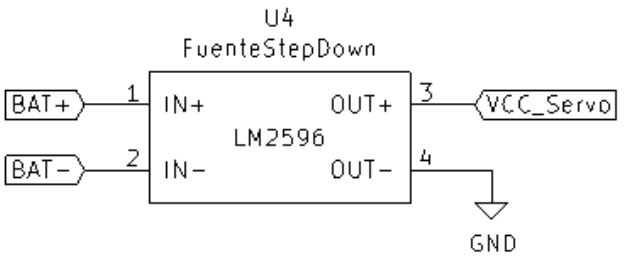

## MICROCONTROLADOR

Al igual que en el control, utilizamos el pic16F886, al cual le colocamos un capacitor de tantalio de 100nF en su alimentación para eliminar posibles ruidos eléctricos.



## ALIMENTACIÓN SERVOMOTOR

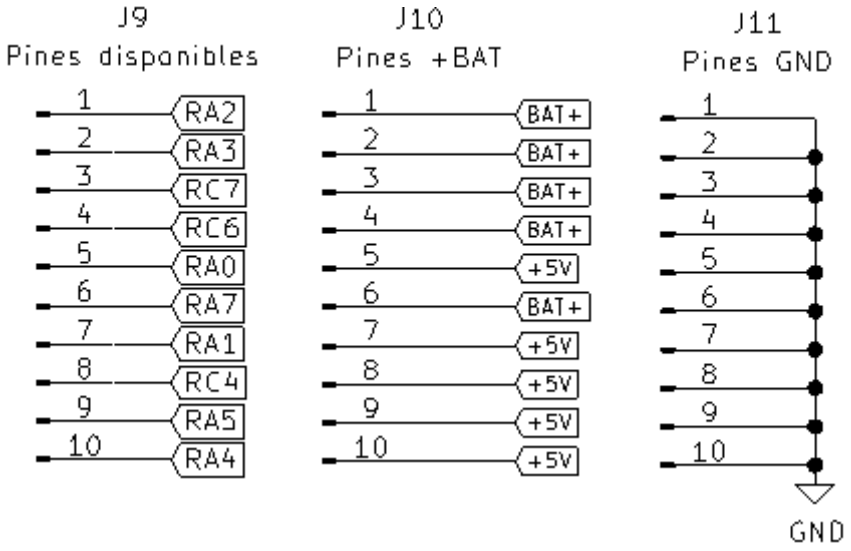
Debido al elevado consumo del servomotor, no nos fue posible alimentarlo con la misma fuente lineal utilizada para la parte de control, por lo tanto, decidimos utilizar una fuente switching de tipo StepDown, ya que el servomotor requiere una tensión de entre 6,8V y 7,2V, y la batería nos entrega entre 7,2V y 8,4V.




## PINES EXTRA DE ALIMENTACIÓN

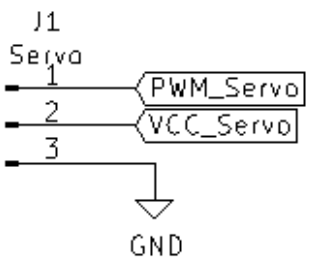
También decidimos colocar tiras de pines adicionales para los pines no utilizados en el PIC, pines conectados a la tensión de 5V, alimentación proveniente de la batería, y pines de conexión a tierra (GND).

Esto nos proporciona flexibilidad para futuras mejoras o expansiones en el auto. Los pines de 5V y GND adicionales permiten conectar nuevos dispositivos o módulos que puedan requerir alimentación o conexión a tierra.



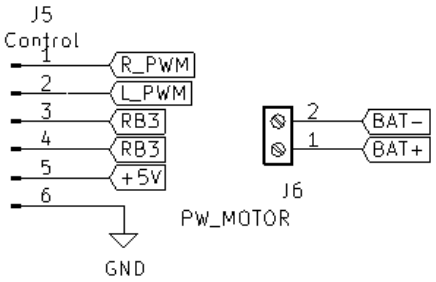

## PINES SERVOMOTOR

Estos pines fueron colocados para no soldar directamente sobre la placa el servomotor, y poder retirarlo fácilmente en caso de realizar alguna modificación.



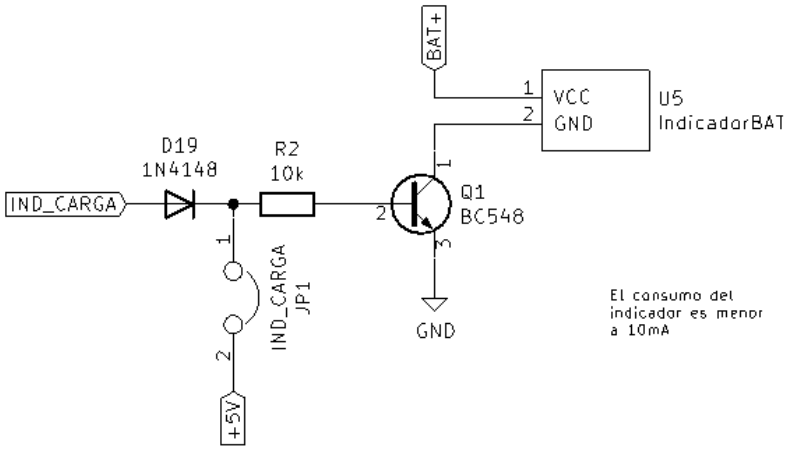
## PINES MÓDULO PUENTE H

Estos pines son utilizados para conectar la parte de control del módulo de puente H, cómo así también hemos agregado bornes de potencia para la alimentación del motor que se conecta a través del módulo.



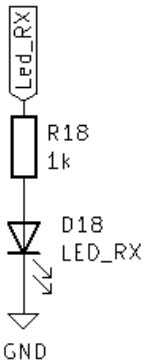

## INDICADOR DE BATERÍA

El siguiente circuito es el encargado de controlar el encendido del módulo indicador de batería, el mismo consta de un transistor, el cuál dependiendo del estado de IND_CARGA entrará en conducción o no, provocando el encendido del indicador. Además el circuito incorpora un jumper, utilizado en caso de que se quiera dejar siempre encendido el display, y un diodo de protección para la entrada del microcontrolador D19.



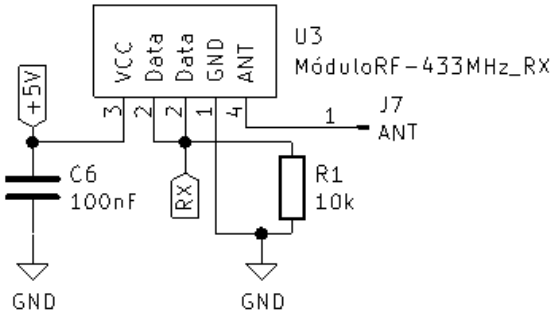

## LED RX

La placa incorpora un led, el mismo indica si la información del joystick está siendo recibida por el auto.



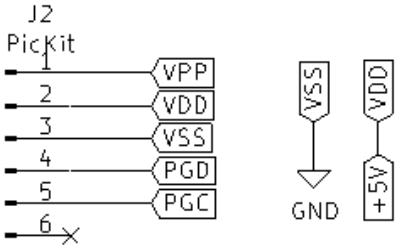
## MÓDULO RECEPTOR

Para el correcto funcionamiento del módulo receptor, se le debe colocar un capacitor de tantalio de 100nF en su alimentación, además para que el microcontrolador interprete bien la información recibida, el pin de data debe ir conectado a una resistencia de PullDown. En cuánto al largo de la antena, utilizamos una de igual longitud a la del joystick.



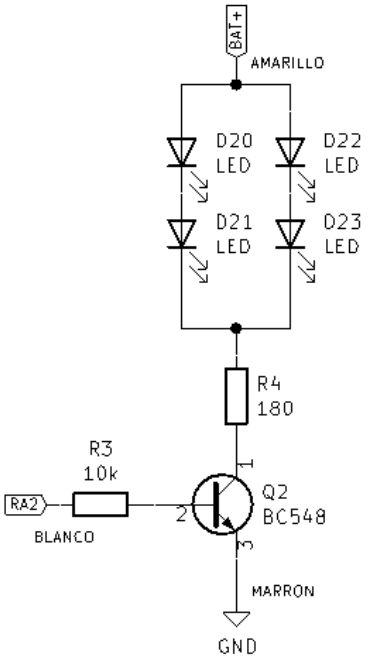

## PINES PICKIT

Para facilitar la carga de nuevos programas al microcontrolador, decidimos agregarle unos pines exclusivos para el programador PicKit.




## LUCES NITRO

El siguiente bloque, se encuentra en el interior de los caños de escape del auto, y es el encargado de encender y controlar las luces de nitro. El cable de señal, es tomado directamente desde uno de los pines disponibles (RA2).



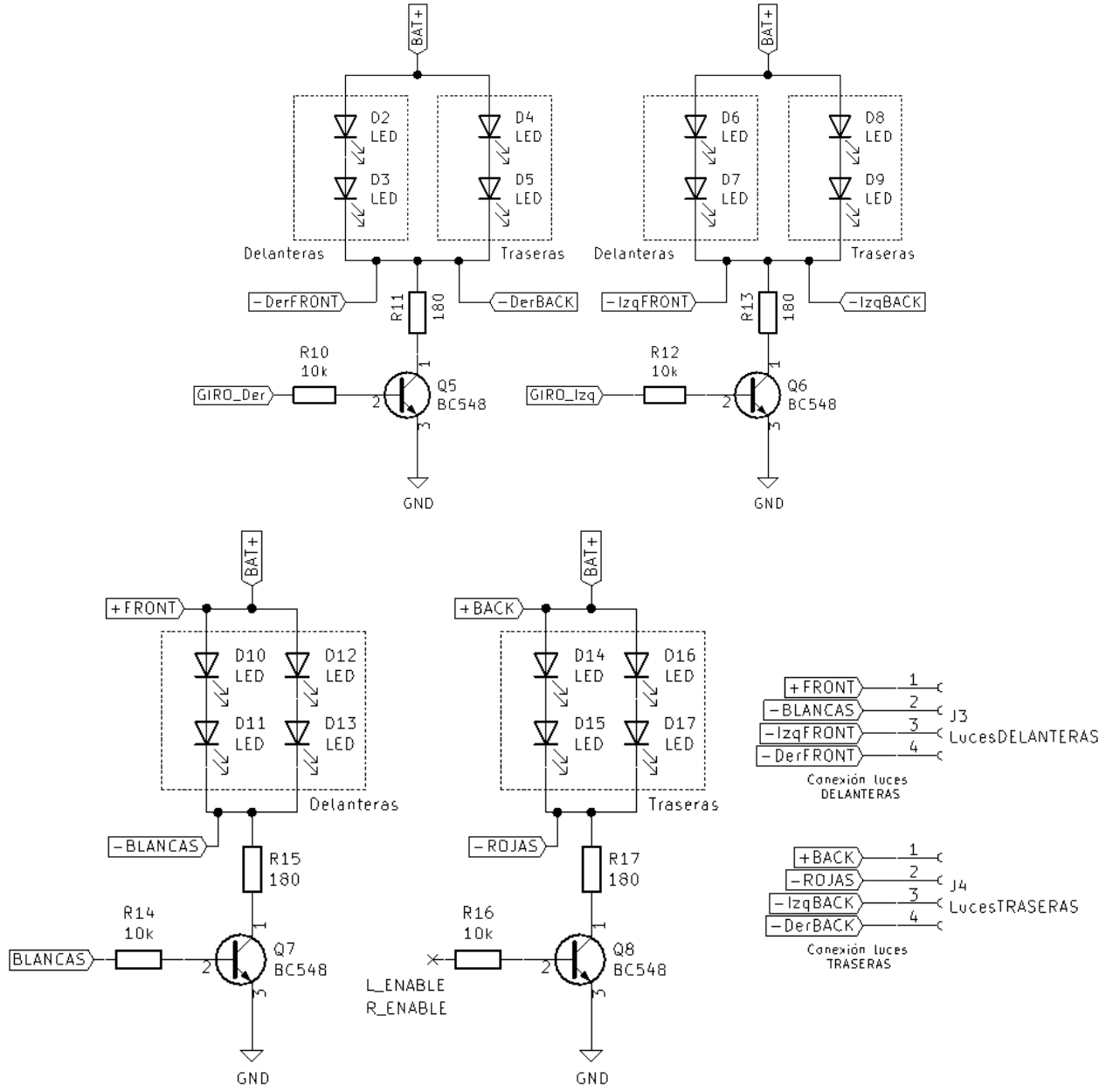
## LUCES

En esta sección, se encuentra el bloque de control de las luces del vehículo, organizado en diferentes grupos para una gestión eficiente de la energía y el funcionamiento óptimo. Están agrupadas en dos grupos de 2 leds cada uno por transistor, para así lograr el menor consumo y la máxima eficiencia en su funcionamiento.

Q5 y Q6: Estos transistores se encargan de controlar las luces de giro, tanto traseras como delanteras, del lado derecho e izquierdo respectivamente.

Q7: Este transistor controla las luces blancas delanteras del vehículo. Estas luces se activan mediante una palanca ubicada en el joystick.

Q8: Este transistor se encarga de controlar las luces rojas traseras, las cuales se activan cuando el vehículo se encuentra en marcha atrás.




Además, dispone de pines para la rápida conexión de ambas luces.

## PCB

El proceso de diseño y fabricación de la PCB destinada al auto implicó enfrentarse a un desafío clave: optimizar al máximo el espacio disponible en una placa de dimensiones reducidas, con medidas de 115 mm de largo por 80 mm de ancho. Nuestra principal visión era crear una placa que permitiera la conexión y desconexión sin necesidad de soldadura de todos los módulos o funcionalidades adicionales que pudieran agregarse al auto.

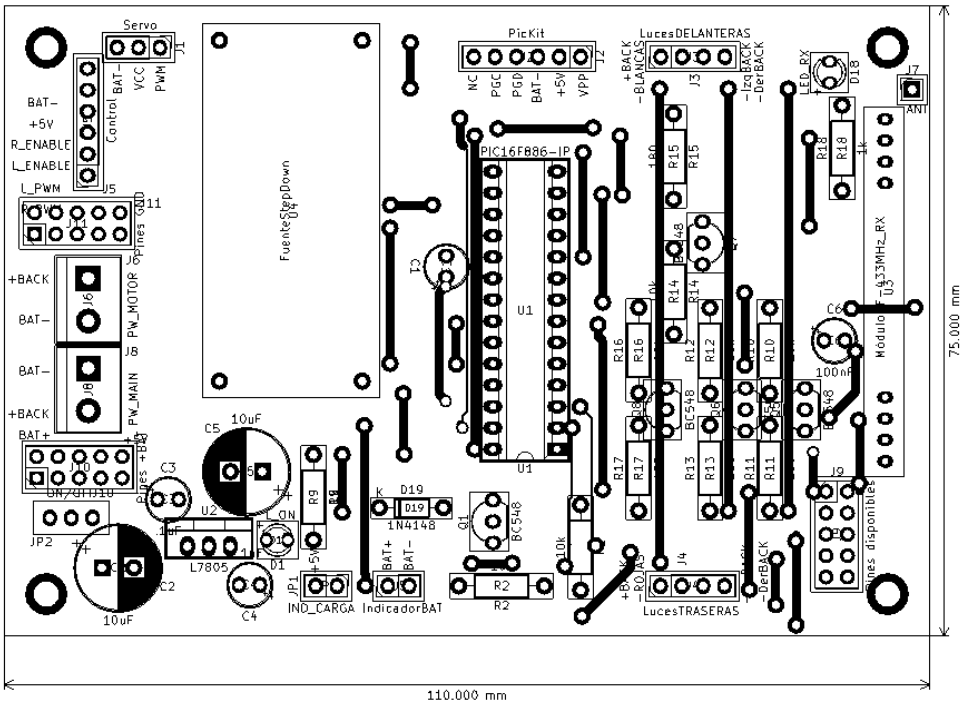
Para lograr esta versatilidad, incorporamos pines de conexión para cada uno de los accesorios, tales como luces, display indicador de batería, entre otros. Esto permitió una conexión fácil y rápida de estos elementos, facilitando el proceso de instalación y transporte del vehículo.

Uno de los desafíos clave fue asegurar que la antena dispusiera de un conector de fácil remoción, permitiendo un montaje ágil y la posibilidad de guardar y transportar el auto con mayor comodidad.

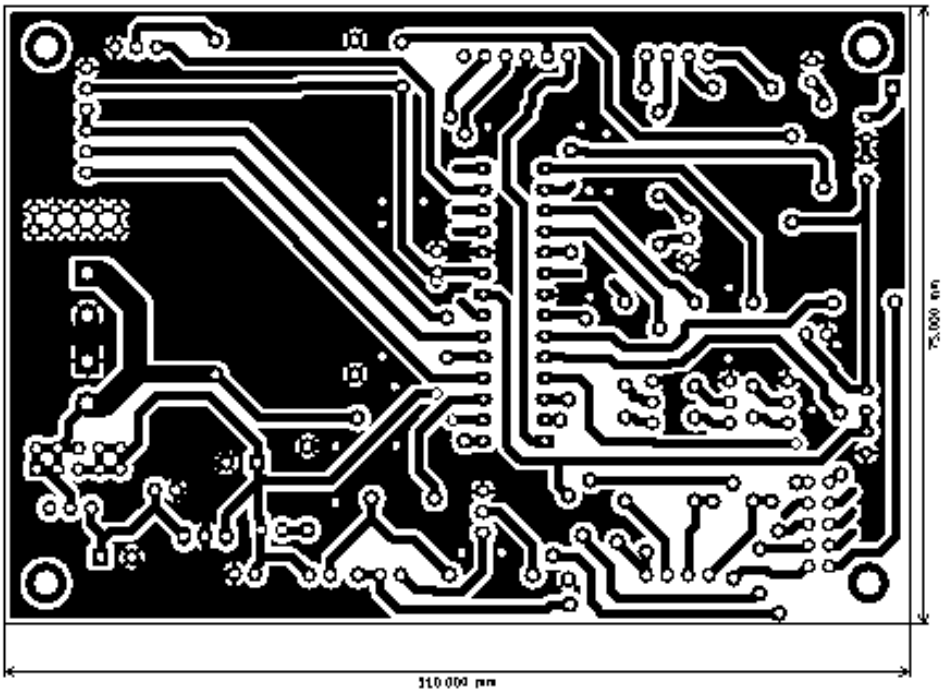
Además, para optimizar aún más el espacio, integramos la fuente StepDown que alimentará el servomotor en la misma PCB. Esta decisión fue motivada por la necesidad de minimizar la conexión de componentes entre sí, manteniendo un diseño compacto y funcional.

Considerando el montaje en el chasis del auto, incorporamos estratégicamente 4 agujeros en los extremos de la placa.

**Vista frontal**

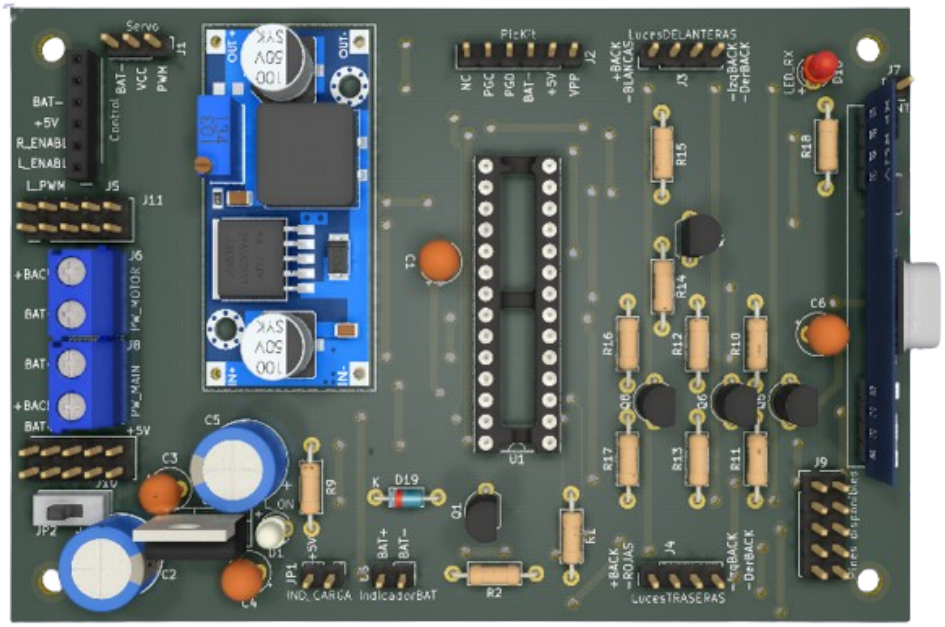


**Vista trasera**

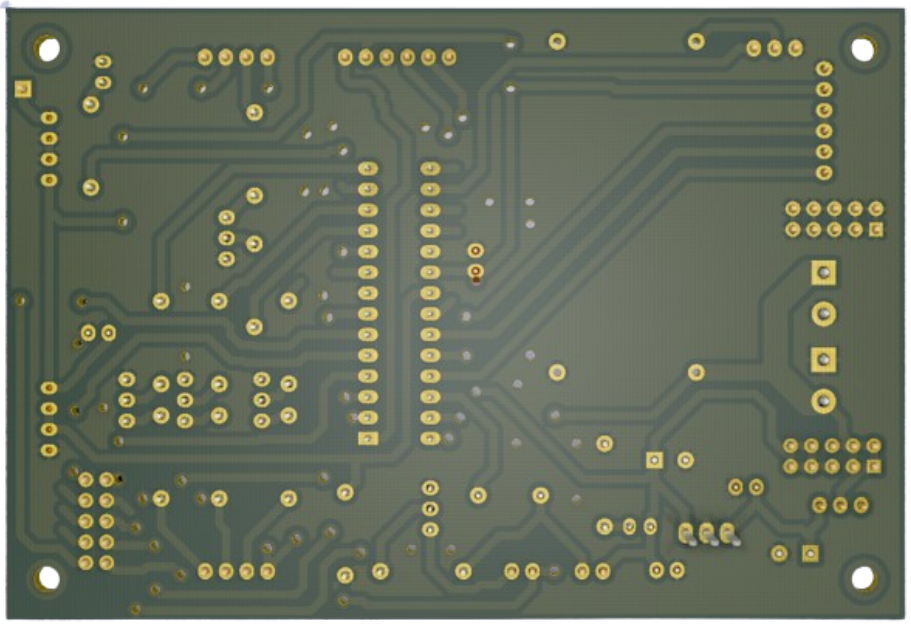



**Renders**

**Frontal:**

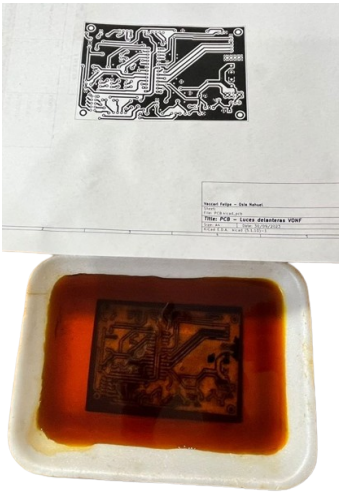


**Trasera:**

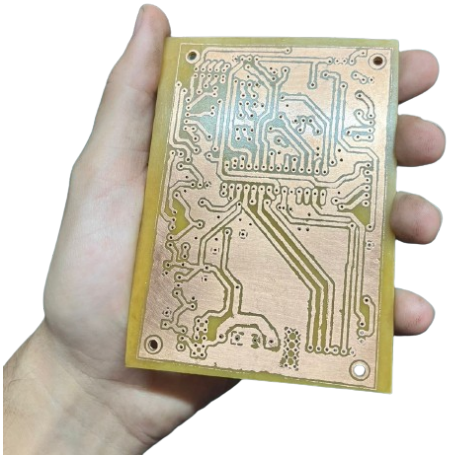


**Proceso de fabricación de la PCB**

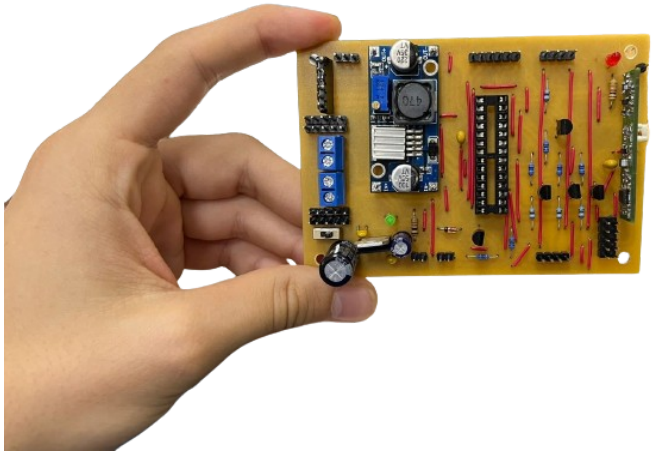
Para fabricar la pcb seguimos los mismos pasos comentados anteriormente para la fabricación de la placa del joystick.



Colocación de placa virgen en ácido



Resultado



Placa con componentes soldados

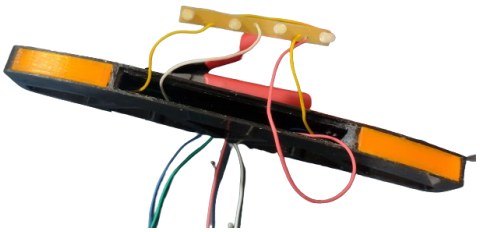
Montaje de la placa en el auto

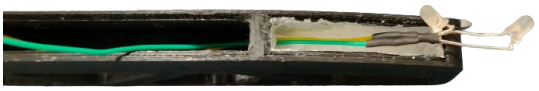
## Extras

A continuación, vamos a explicar el proceso de montaje y armado de todos los extras como la iluminación, el nitro, la antena, entre otros.

**Armado de luces:**

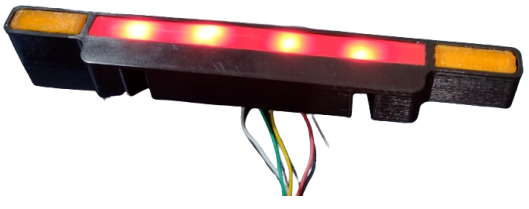
El proceso de armado de las luces implicó un enfoque detallado para integrarlas de manera óptima al chasis del auto RC. Nuestro objetivo principal fue lograr una integración fluida: colocamos las luces delanteras dentro del paragolpes delantero y las traseras en el alerón trasero, diseñando ambas piezas según las medidas y cantidades de diodos LED previstos para cada una.



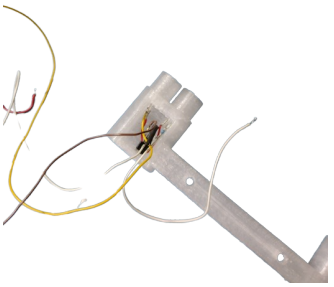


Un aspecto crucial para nosotros fue lograr una difusión uniforme de la luz para evitar que los LEDs se vieran como puntos individuales y, en su lugar, brindaran una iluminación más homogénea. Para alcanzar este efecto, nos enfrentamos al desafío de difuminar la luz de los LEDs. Al no conseguir diodos LED difusos, tuvimos que lijar individualmente cada LED. Además, colocamos papel aluminio en el interior de estos compartimentos para obtener el efecto deseado.

Aunque el resultado final nos satisface, notamos que las luces delanteras podrían mejorar en su intensidad, especialmente para simular luces altas de manera más efectiva. Esta es una área que hubiéramos deseado mejorar para lograr un mayor realismo en la función de las luces altas del auto.

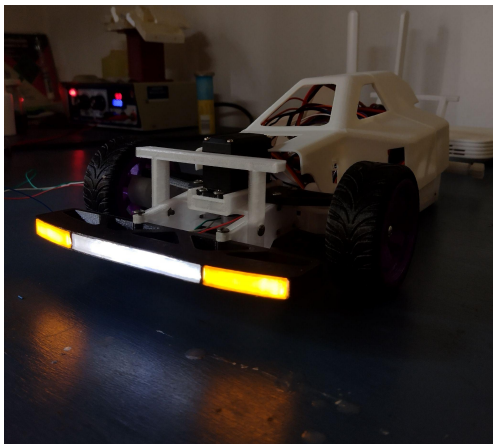


**Armado de caños de escape:**

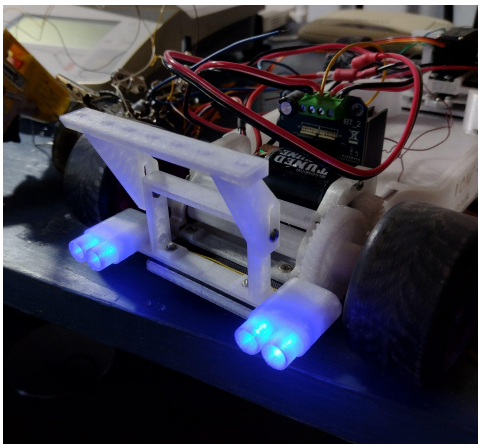


Para recrear el efecto de aumento de potencia al activar el botón de nitro, decidimos incorporar una simulación en el caño de escape del auto RC. En el interior de esta pieza, colocamos 4 LEDs azules. Estos LEDs fueron posicionados para simular visualmente el efecto de un aumento de energía al activar el nitro.

Además de los LEDs, diseñamos un compartimento específico dentro de la misma pieza plástica para alojar el transistor responsable de controlar estas luces, junto con su respectiva resistencia limitadora.



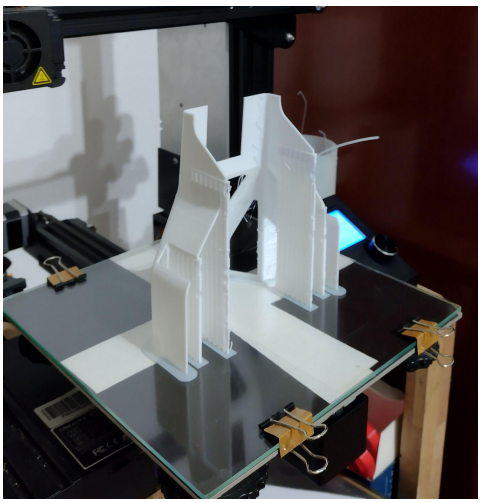
Colocación de luces delanteras



Caños de escape encendidos

**Chasis superior**

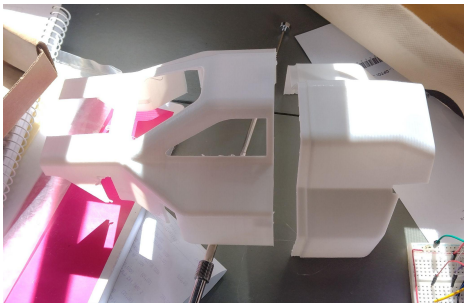
Para el diseño del chasis superior, optamos por una estética simple que permitiera visualizar el interior del auto RC, exhibiendo la electrónica y ofreciendo una apariencia de auto deportivo.



Decidimos incorporar una cámara GoPro en la parte superior del chasis. Esta cámara nos proporciona la capacidad de transmitir vídeo en vivo a través de una conexión WiFi, con un alcance aproximado de 10 metros. Utilizamos esta funcionalidad para visualizar en nuestros dispositivos móviles lo que capta la cámara, lo que agregó un atractivo adicional al proyecto.

En el frente del chasis, instalamos el indicador de batería y añadimos ventanas plásticas que permiten visualizar el interior del vehículo. También, aplicamos stickers creados por nosotros, de reconocidos patrocinadores de fórmula 1 para darle un aspecto más auténtico y realista.

Cómo era de esperar, debido al gran tamaño del auto, imprimimos el chasis en dos mitades, que posteriormente las unimos. A continuación se observan imágenes del diseño y la postproducción luego de la impresión 3D del mismo.



Impresión del chasis


Chasis terminado

**Mejoras en la estructura**

Inicialmente, tanto el eje trasero como los ejes de las ruedas delanteras del auto estaban fabricados completamente con PETG, el material utilizado para imprimir la estructura del vehículo. Sin embargo, un incidente durante la exposición del día jueves 30/11 provocó la rotura de una de las ruedas traseras del auto, lo que nos imposibilitó exhibirlo en funcionamiento pleno durante ese mismo día. Ante esta urgente necesidad de reparación, decidimos reforzar ambos ejes con piezas metálicas en menos de 24 horas, ya que debíamos volver a exponer el viernes 1/12 con el auto en pleno funcionamiento.

La solución planteada para mejorar la resistencia del eje trasero fue instalar una varilla roscada de 4 mm a lo largo de su interior. Aunque esta adición incrementó el peso total del auto, su impacto se reflejó significativamente en la rigidez y durabilidad general del vehículo.

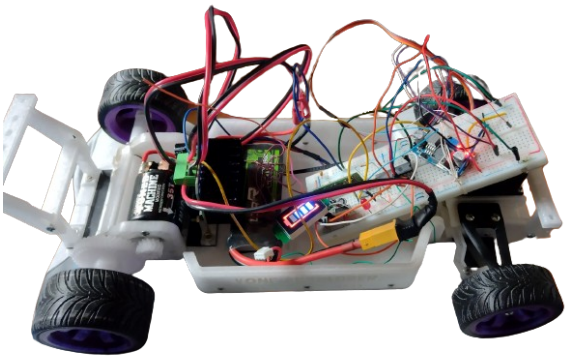
En cuanto a las ruedas, implementamos un método de sujeción más robusto utilizando tuercas con arandelas grower.

Para reforzar los ejes de las ruedas delanteras, optamos por añadir tornillos M3 de 35 mm de longitud, los cuales aseguramos desde el exterior.

Eje nuevo

**Colocación de la electrónica dentro del auto**

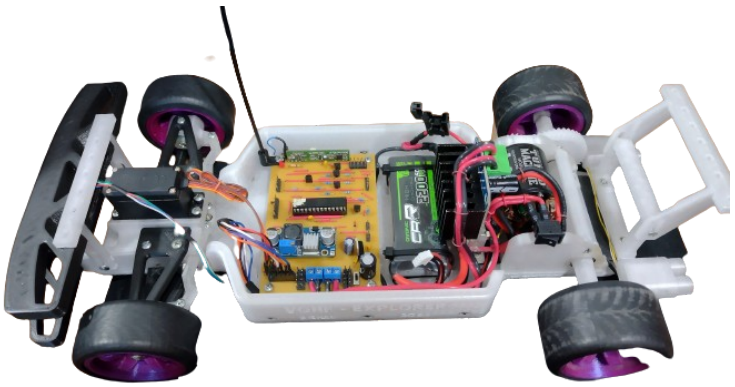
El ensamblaje de esta parte del proceso resultó ser uno de los desafíos más complejos debido a la gran cantidad de componentes que debíamos integrar en el espacio reducido del auto. Además, debíamos tener especial cuidado con la elevada corriente de consumo del motor, ya que conexiones flojas o mal realizadas podrían ocasionar caídas de tensión significativas en los conectores o conductores, aumentando el riesgo de cortocircuitos e incluso incendios en el vehículo.



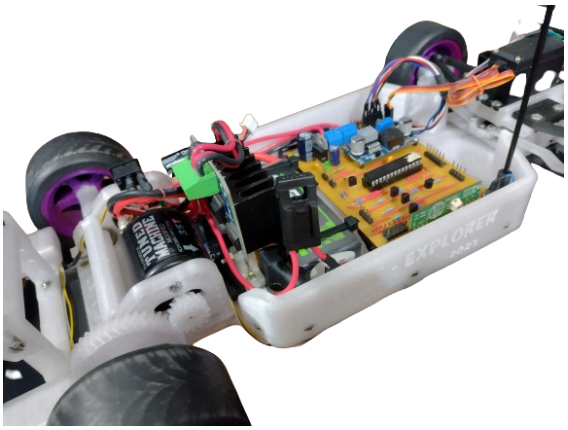
Para mitigar cualquier posibilidad de fallo y en caso de que la protección de la batería no funcionara, decidimos instalar un portafusibles, similar a los utilizados en vehículos convencionales, con un fusible de 10A. Además de incorporar un interterruptor de corte general.

Por otro lado, optamos por mantener la batería fija en el vehículo para evitar manipularla constantemente. Para facilitar su carga, extendimos el cable de carga, permitiendo realizar esta operación sin tener que retirar la misma del auto.

Además, nos esforzamos por realizar un cableado ordenado y cuidadoso, seleccionando la sección adecuada de los conductores para aquellos cables por donde circula una mayor corriente.



Interruptor



Portafusibles

**Cambio de servomotor**

Luego de haber mejorado el eje con la varilla roscada, el viernes al mediodía, decidimos probar el auto. Funcionó correctamente durante unos pocos minutos, pero para nuestra sorpresa, la dirección dejó de responder de manera inesperada y sin una causa aparente. A pesar de que el servomotor estaba alimentado y recibía pulsos, se negaba a funcionar. Intentamos cargar nuevamente el programa en el auto y probamos el servo por separado fuera del vehículo para descartar fallas en la placa, pero no obtuvimos respuesta.

Dado que nos encontrábamos a solo cinco horas de la exposición y el auto no funcionaba, nos vimos obligados a tomar medidas rápidas. Decidimos desmontar el servo para encontrar la causa del problema. Al revisar la placa de control del servomotor, todo parecía estar bien, entonces decidimos desarmar por completo el motor, al desarmarlo descubrimos que las escobillas del mismo estaban prácticamente fundidas debido a la temperatura. Parecía que el motor se había forzado, causando la rotura de las escobillas. Esta situación imposibilitaba una reparación rápida.

Ante esta circunstancia, optamos por adquirir un servo nuevo. Afortunadamente, logramos encontrar uno a tiempo y lo instalamos justo antes de la exposición, permitiéndonos presentar el auto en pleno funcionamiento durante el evento.

Sin duda, la reparación del eje trasero y el cambio del servomotor representaron un verdadero desafío, especialmente porque nos encontrábamos a pocas horas de la exposición. Pero la sensación de alivio y alegría al encontrar un reemplazo a tiempo fue indescriptible. Sentir el alivio al instalar el nuevo servo y ver al auto funcionar nuevamente, listo para ser presentado en la exposición, fue un momento realmente gratificante.

**Datos relevantes del auto**

El auto, junto con la GoPro, tiene un peso total de 1,5 kg, lo que representa un vehículo considerablemente pesado debido a su tamaño y la abundancia de componentes. Por otro lado, el joystick tiene un peso total de 250 g. En cuanto al tiempo requerido para la impresión, incluyendo el vehículo, el chasis, los accesorios, los repuestos y el control, nos llevó más de 200 horas de trabajo. Esto equivaldría a más de 8 días consecutivos de funcionamiento de la impresora 3D.

En términos de filamento 3D, estimamos haber utilizado al menos 1,5 kg para completar todo el proyecto en general.

**Proyecto terminado**





## COSTOS FINALES

Estos costos, no tienen en cuenta algunos gastos extras, de circuitos descartados, componentes quemados, perdidos, etc. Los precios en pesos se actualizan a la fecha 1/12/23.

| GASTOS FINALES | Pesos ARG $ | USD $ |
|---|---|---|
| Módulo 433MHz | 1800 | 1,8 |
| Tornillería | 2000 | 2,00 |
| Pic 16F886 (x2) | 40000 | 40,00 |
| Motor brushed 35T | 14000 | 14,00 |
| Servo motor | 14000 | 14,00 |
| Joysticks (2) | 5000 | 5,00 |
| Puente H | 18000 | 18,00 |
| Bateria 2s 2200mAh | 18000 | 18,00 |
| Baterías 3,7v 650mAh x2 | 14000 | 14,00 |
| Cargador litio 2s | 12000 | 12,00 |
| Rodamientos | 16000 | 16,00 |
| Indicador nivel de batería | 5000 | 5 |
| Zócalo dip 2x10 | 600 | 0,60 |
| Placa virgen 10x20cm | 3500 | 3,50 |
| Placa virgen. 10x10cm | 2000 | 2,00 |
| Zócalo dip 2x9 | 600 | 0,60 |
| Zócalo dip 2x14 (x2) | 1200 | 1,20 |
| 600g PLA | 9000 | 9,00 |
| 700g PETG | 9800 | 9,80 |
| 300g TPU | 8000 | 8,00 |
| Ácido Férrico 500CC | 4000 | 4,00 |
| Tira de pines | 1000 | 1,00 |
| lm7805 x2 | 3000 | 3,00 |
| capacitores 100nF x4 | 400 | 0,40 |
| Cables 1mm | 1060 | 1,00 |
| zócalo para pic 16f886 | 300 | 0,30 |
| micro interruptor | 800 | 0,80 |
| 4 leds 3mm azules (alto brillo) | 800 | 0,80 |
| TOTAL | 205860 | 205,8 |

# Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a todas las personas que han contribuido significativamente al desarrollo y éxito de este proyecto. Sus valiosos aportes, apoyo constante y orientación han sido pilares fundamentales en cada fase de este emocionante proceso. Específicamente, deseamos reconocer y expresar nuestro agradecimiento a:

**Hernán Fiscella:** Por su enorme ayuda en la obtención de componentes fundamentales que de otro modo habrían sido imposibles de adquirir. Su disposición para brindarnos espacio en su materia y su asistencia en la resolución de inquietudes durante el desarrollo del proyecto fueron de gran valor.

**Gustavo García:** Por su asesoramiento y recomendaciones en la selección de componentes, diseño de circuitos y valiosa ayuda en la resolución de problemas.

**Martín Gracia:** Por su constante apoyo, orientación y aliento incondicional a lo largo de todo el proyecto. Su motivación fue un impulso vital para nosotros.

**José Luis Rodríguez:** Por su gran apoyo y por facilitarnos los microcontroladores esenciales para la realización del proyecto.

**Mauricio Martínez:** Por su asistencia en la resolución de problemas de programación y su experiencia, que fue esencial para superar obstáculos técnicos de manera efectiva.

**Mauro Antivero:** Por su orientación en la selección de motores, baterías, puentes H, recomendaciones de impresión 3D, programación del servo, y consejos vitales para el desarrollo del proyecto.

También queremos expresar nuestra profunda gratitud a MiamiGo, una destacada empresa de Courier Internacional con sede en Rosario. Su colaboración fue esencial para importar los componentes fundamentales utilizados en nuestro proyecto desde el extranjero. MiamiGo se distingue por su servicio rápido y eficiente, ofreciendo una conexión directa entre Miami y Rosario.

Por último, pero no menos importante, queremos expresar nuestro mayor agradecimiento a nuestras familias, quienes desde el inicio del proyecto nos brindaron un apoyo incondicional en cada decisión tomada. A su vez, extendemos nuestro agradecimiento a todos los demás profesores del laboratorio de electrónica, cuyo apoyo indirecto y disposición para ayudarnos ha sido de gran valor. ¡Gracias!.

# Conclusiones

Concluimos este fiel informe del desarrollo de nuestro proyecto de sexto año, el cual hemos realizado con mucho esfuerzo, cariño, y dedicación.

VONF Explorer, ya sea en lo económico, en lo académico, y en lo personal, ha conllevado una gran inversión. El tiempo fuera de casa, en las profundas investigaciones de técnicas, componentes, tecnologías y en tantas otras cosas. Este proyecto nos ayudó a consolidar todos los conocimientos adquiridos a lo largo de nuestra carrera, dejó de ser un proyecto estudiantil y comenzó a ser una meta personal. Realmente nos sirvió muchísimo, ya que todo lo que requirió este proyecto, es perfectamente aplicable en otros, cada uno de los conocimientos adquiridos, ya sea respecto el diseño de circuitos y PCB´s, en la programación de un PIC ajeno al dado en la carrera, diseño de piezas 3D personalizadas y específicas, sistemas de comunicaciones, lecturas exhaustivas en datasheets, y otros como búsqueda de componentes y proveedores, o el desarme de electrodomesticos viejos.

También mejoramos destacablemente nuestras habilidades blandas, en lo que respecta a la creatividad, hemos diseñado nosotros partes esenciales del explorador que eran particulares y lo diferenciaban de un auto RC común y corriente, ya sea su chasis, su control lumínico, la modalidad de turbo, lector de batería, el chasis del control, y demás detalles de índole eléctronico. Hemos desarrollado la paciencia, a la hora de esperar componentes, hemos aprendido a organizarnos en tiempo y espacio, con las herramientas dadas, buscando sacarle provecho al máximo a todo lo brindado por la escuela. Nos hemos complementado muy eficazmente a la hora de trabajar en equipo, dividiendo las áreas de trabajo, y tareas a realizar por cada uno.

Por otro lado nos hemos enfrentado a diversos problemas lo cuales suponían un desgaste mental, debido a que estos problemas, fueron totalmente impredecibles, pero gracias al margen de tiempo, y nuestra correcta organización, no significaron un gran inconveniente a la hora de la exposición de VONF.

Cerramos este informe muy contentos de haber superado nuestras expectativas, en todas las áreas, y con la certeza de haber crecido muchísimo cómo técnicos y personas, este informe a la fecha actual del 1/12/23, estará desactualizado, debido a que VONF, continuará en desarrollo constante, corrigiendo cualquier defecto presente y agregando todas las mejoras posibles dadas en lo que respecta la tecnología actual.