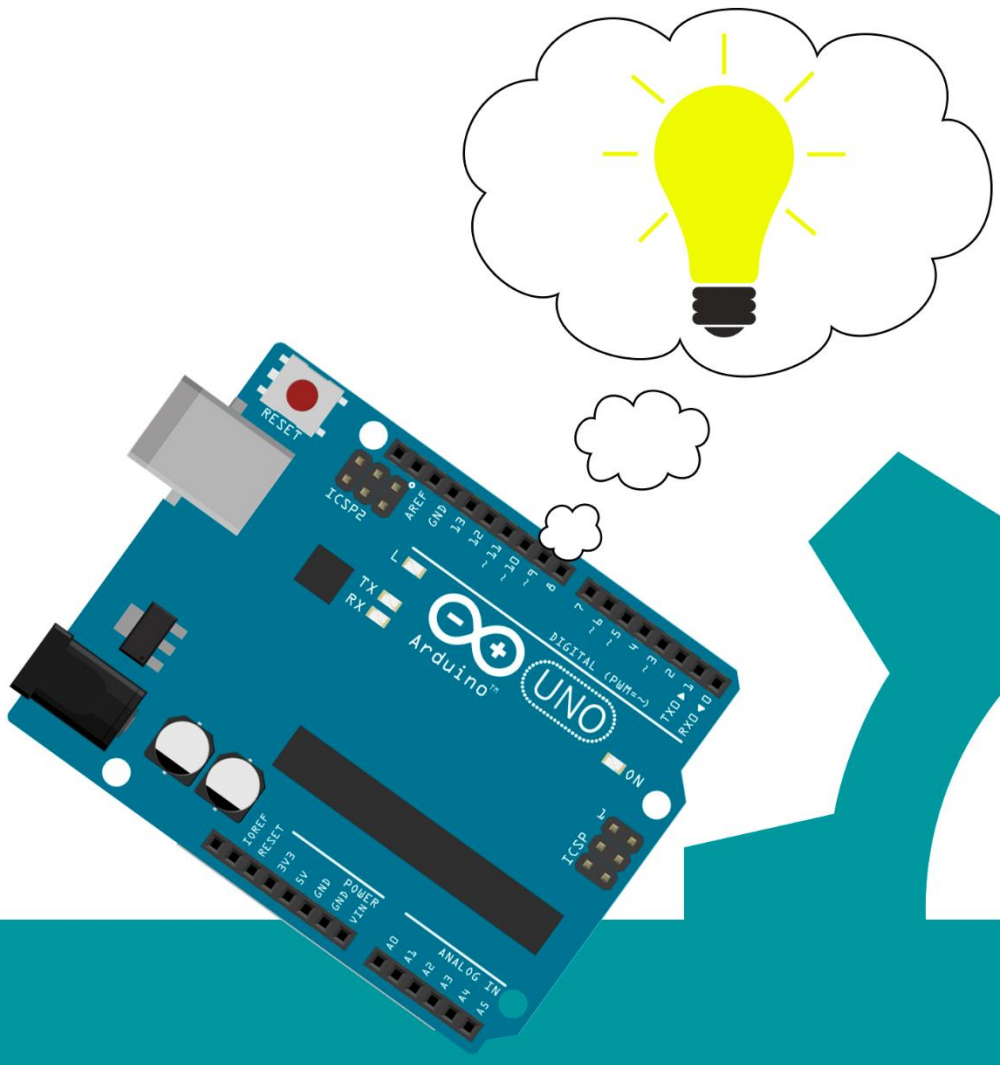
CURSO BASICO DE ARDUINO



Misael Saenz Flores

MECATRÓNICA LATAM

El autor del libro de Curso Básico de Arduino: Misael Saenz Flores

Proyectos y textos elaborados por: Misael Saenz Flores

Corrección de texto:

- Mecatrónica LATAM
    - Méndez Goerne Erick Oswaldo
    - Montecillo Torres Magnolia Angelina.

Diseño y dirección de arte: Misael Saenz Flores e Imágenes libres de la red pertenecientes a sus respectivos creadores

MeArm desarrollado por Ben Gracy

Circuitos elaborados por: Misael Saenz Flores

Software para la elaboración de circuitos:

Fritzing - http://fritzing.org

Obra basada en:

Curso Básico de Arduino en YouTube - https://goo.gl/L5FJGx

En el capítulo 1 parte de este está basado en el texto de Fco. Javier C.

Agradecimiento:

Quiero dar las gracias a toda la comunidad de Arduino por la contribución que realiza constantemente, gracias a ello también se pudo realizar esta obra, también doy gracias a las personas de mi vida personal que me apoyaron durante todo este tiempo en que se escribió esta obra.

INDICE

Nota: Al final de cada capítulo o entre los capítulos habrá un icono del sitio web YouTube, esto indica un video tutorial como apoyo al capítulo correspondiente.

INTRODUCCIÓN

El Capítulo 1 describe una introducción a la programación y conocer los antecedentes necesarios para poder aplicarlos a los siguientes capítulos, así mismo se ven conceptos básicos sobre qué son las variables, los tipos de datos, condicionales, ciclos, etc., esto se realizará en lenguaje C ya que es muy parecido al lenguaje que usa la plataforma Arduino.

El Capítulo 2 inicia con la descarga e instalación de los softwares necesarios para la programación de nuestra tarjeta Arduino, mediante ilustraciones gráficas se seguirá paso a paso la descarga de cada uno de los programas, esto permite al lector una guía simple y completa.

El Capítulo 3 presenta una breve explicación sobre electrónica básica y las tarjetas de desarrollo, específicamente sobre la tarjeta Arduino UNO y sus características, además se proporcionan consejos muy útiles para el correcto funcionamiento de ésta.

El Capítulo 4 inicia con ejercicios básicos para poder comprender el funcionamiento de nuestra tarjeta de desarrollo Arduino. También cabe recalcar que es indispensable, pero no necesario disponer de alguna tarjeta de desarrollo ya sean las versiones Arduino UNO, Arduino Duemilanove, Arduino NANO, Arduino Mega, etc., también es necesario seguir en orden cada una de las prácticas para poder entender los conceptos y prácticas que se muestran más adelante, ya que cada vez serán más extensas, pero no muy difíciles de entender.

El Capítulo 5 presenta algunos proyectos donde se pondrá en práctica lo aprendido en los capítulos anteriores y así poder completar con gran éxito los proyectos necesarios, para su trabajo y/o escuela.

CAPÍTULO 1

INTRODUCCIÓN A LA PROGRAMACIÓN

1.0 ELEMENTOS DEL LENGUAJE

En este capítulo se verán los elementos que aporta el lenguaje C (caracteres, secuencias de escape, tipos de datos, operadores, etc.) para escribir un programa. Considere este capítulo como soporte para el resto de los capítulos.

1.1 CARACTERES EN C

Los caracteres del lenguaje en C pueden agruparse en letras, dígitos, espacios en blanco, caracteres especiales, signos de puntuación y secuencias de escape.

LETRAS, DÍGITOS, ETC.

Estos caracteres son utilizados para formar las *constantes*, los *identificadores* y las *palabras clave* del lenguaje C. Son los siguientes:

- Las letras mayúsculas del alfabeto inglés:
  - A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- Las letras minúsculas del alfabeto inglés:
  - a b c d e f g h i j k l m n o p q r s t u v w x y z
- Dígitos enteros
  - 0 1 2 3 4 5 6 7 8 9 …
- Carácter de subrayado “_”

El compilador trata las letras mayúsculas y minúsculas como caracteres diferentes. Por ejemplo los identificadores de la constante de Euler *e* y *E* son diferentes.

ESPACIOS EN BLANCO

Los espacios en blanco, el tabulador (todos los tipos), avance de página y nueva línea, son caracteres denominados *en blanco*, debido a que la labor que desempeñan es la misma que la del espacio en blanco: actuar como separadores, lo cual permite escribir un programa mucho más limpio y legible. Por ejemplo:

```
main() { printf(“Hola Mundo.\n”); }
```

Puede escribirse de una forma más legible así:

```
main(){
      printf("Hola Mundo.\n");
}
```

Los espacios en blanco en exceso son ignorados.

CARACTERES ESPECIALES Y SIGNOS DE PUNTUACIÓN

Este grupo de caracteres se utilizan de diferentes formas, para indicar que un identificador es una función o una matriz; para especificar una determinada operación aritmética lógica o de relación, etc. Son los siguientes:

```
, . ; : ¿ ' " ( ) [ ] { } < ! | / \ ~ + # % & ^ * - =
```

1.2 TIPO DE DATOS

Veamos el siguiente programa que lleva a cabo una operación aritmética la cual realiza una suma de dos valores:

```
variable1 = 80;

variable2 = 10;

resultado = variable1 + variable2;
```

Para que el compilador reconozca esta operación es necesario especificar previamente el tipo de variable para cada uno de los operandos que intervienen en la misma, así como el tipo de variable del resultado. Para ello escribiremos una línea como la siguiente:

```
int variable1, variable2, resultado;

variable1 = 80;

dato2 = 10;

variable2 = variable1 + variable2;
```

La declaración anterior le dice al compilador que *variable1, variable2* y *resultado* son del tipo entero (**int**). Observe que se puede declarar más de una variable del mismo tipo empleando una lista separada por comas.

Los tipos de datos se clasifican como: tipos *primitivos* y tipos *derivados*.

TIPOS PRIMITIVOS

Se les llama primitivos porque están definidos por el compilador. Hay siete tipos primitivos de datos que se clasifican en: variables tipo enteros y variables tipo reales.

Tipos enteros: **char, short, int, long** y **enum.**

Tipos reales: **float** y **double.**

Cada tipo primitivo abarca un rango diferente de los valores positivos y negativos.

El tipo de datos que se seleccione para declarar las variables de un determinado programa dependerá del rango y del tipo de valores que vayan a almacenar cada una de ellas, así como si son enteros o fracciones.

CHAR

El tipo **char** declara datos enteros entre -128 y +127. Un tipo **char** se define como un conjunto de 8 bits, de los cuales uno es para especificar el signo y el resto para el valor; dicho conjunto de bits recibe el nombre de *byte*. El siguiente ejemplo declara la variable *b* tipo **char** y le asigna el valor inicial de 0. Es recomendable iniciar todas las variables que se declaren.

```
char t = 0;
```

Los valores de 0 a 127 corresponden con los 128 primeros caracteres de los códigos internacionales ASCII, ANSI o UNICODE empleados para la representación de caracteres.

El siguiente ejemplo declara la variable *car* del tipo **char** a la que se le asigna el carácter *'a'* como valor inicial. Las cuatro declaraciones siguientes son idénticas:

```
char caracter = 't';

char caracter = 116; // la 't' es el decimal 116 en ASCII

char caracter = 0x74; // la 't' es en hexadecimal 0074

char caracter = 0164; // la 't' en octal es 0164
```

SHORT

El tipo **short**, abreviatura de **signed short int,** permite declarar datos enteros comprendidos entre -32768 y +32767. Un valor **short** se define como un dato de 16 bits de longitud, independientemente de la plataforma utilizada. El siguiente ejemplo declara *x* y *y* como variables enteras del tipo **short:**

```
short x = 0, y = 0;
```

INT

El tipo **int**, abreviatura de **signed int** permite declarar datos enteros comprendidos entre -2147483647 y +2147483647. Un valor **int** se define como un dato de 32 bits de longitud. El siguiente ejemplo declara e inicia tres variables *a, b* y *c*, de tipo **int.**

```
int a = 2000;

int n = -30;

int c = 0XF003 // valor en hexadecimal
```

En general, el uso de enteros de cualquier tipo produce un código compacto y rápido.

LONG

El tipo **long** se utiliza para declarar datos enteros comprendidos entre los valores -214748648 y +2147483647. Un valor **long** se define como un dato de 32 bits de longitud. El siguiente ejemplo declara e inicia las variables *a, b* y *c*, del tipo **long.**

```
long a = -1L /* L indica que la constante -1 es long */

long b = 125;

long c = 0x1F00230F; /* valor en hexadecimal */
```

FLOAT

El tipo **float** se utiliza para declarar un dato en coma flotante de 32 bits en el formato IEEE 754. Los datos de tipo **float** almacenan valores con una precisión aproximada de 7 dígitos. Para especificar que una constante es un tipo **float**, hay que añadir al final de su valor la letra ‘f’ o ‘F’. En el siguiente ejemplo se declaran las variables *a, b* y *c*:

```
float a = 3.141592F;

float b = 2.2e-5F /* 2.2e-5 = 2.2 por 10 elevado a -5 */

float c = 2/3.0F; /* 0.666667 */
```

DOUBLE

El tipo **double** se utiliza para declarar un dato en coma flotante de 64 bits. Los datos de tipo **double** almacenan valores con una precisión aproximada de 16 dígitos. Por omisión, una constante es considerada del tipo **double**. En el siguiente ejemplo se declaran las variables *x, y* y *z*:

```
double x = 3.141592; // una constante es double por omisión

double y = 2.2e+8F // 2.2e+5 = 2.2 por 10 elevado a 8

double z = 5/4.0;
```

1.3 COMENTARIOS

Un comentario es un mensaje que se encuentra en el código fuente. Añadiendo comentarios se hace más fácil la comprensión de un programa para el programador, la finalidad de los comentarios es explicar el código fuente o incluir mensajes importantes. El compilador soporta dos tipos de comentarios:

- *Comentario tradicional*. Un comentario tradicional empieza con los caracteres /** y finaliza con */. Estos comentarios pueden ocupar más de una línea, pero no pueden anidarse, y pueden aparecer en cualquier lugar donde se permita aparecer un espacio en blanco. Por ejemplo:
  ```
  /**
   * Esto es un comentario tradicional, que puede ir en la cabecera de las funciones de su programa.
   *
   */
  ```
- *Comentario de una sola línea*. Este tipo de comentario comienza con una doble barra (//) y se extiende hasta el final de la línea. Esto quiere decir que han incorporado algunas características de interés de C++; una de ellas es esta. La siguiente línea muestra un ejemplo:
  ```
  // Agregar aquí el código de iniciación
  //Este es otro comentario de una sola línea
  ```

1.4 DECLARACIÓN DE CONSTANTES SIMBÓLICAS

Declarar una constante simbólica significa decirle al compilador el nombre de la constante y su valor. Esto se hace generalmente antes de la función **main** utilizando la directriz **#define**, cuya sintaxis es así:

```
#define NOMBRE VALOR
```

El siguiente ejemplo declara la constante real *PI* con el valor de 3.14159, la constante de un solo carácter *NL* con el valor '\n' y la constante de caracteres *MENSAJE* con el valor "Pulse una tecla para continuar\n":

```
#define PI 3.14159

#define SALTO '\n'

#define MENSAJE "Pulse una tecla para continuar\n"
```

Observe que no hay un punto y coma después de la declaración. Esto es así, porque una directriz no es una sentencia, sino una orden para el preprocesador. El tipo de una constante es el tipo del valor asignado.

Suele ser habitual escribir el nombre de una constante en mayúsculas.

Otra de las características incorporadas por los compiladores es la palabra reservada **const**. Utilizándola disponemos de una forma adicional para declarar una constante; basta con anteponer el calificador **const** al nombre de la constante seguido del tipo de la misma; si el tipo se omite, se supone **int**. Por ejemplo, la siguiente línea declara la constante real *Pi* con el valor 3.14:

```
const double E = 2.71;
```

Una vez declarada e iniciada una constante, ya no se puede modificar su valor. Por ello, al declararla debes ser iniciada.

1.5 DECLARACIÓN DE UNA VARIABLE

Una variable representa un espacio de memoria para almacenar un valor de un determinado tipo. El valor de una variable, a diferencia de una constante, puede cambiar su valor durante la ejecución de un programa. Para utilizar una variable en un programa, primero hay que declararla. La declaración de una variable consiste en enunciar el nombre de la misma y asociarle un tipo:

```
tipo nombre, nombre,…
```

En el siguiente ejemplo se declaran e inician cuatro variables: una del tipo **char, int, float** y **double**:

```
char c = 't';

main(){

int i = 0;

float f = 0.0F;

double d = 0.0;

//…

}
```

El tipo, primitivo o derivado, determina los valores que puede tomar la variable así como las operaciones que pueden realizarse con ella. Los operadores serán expuestos más adelante.

En el ejemplo anterior se puede observar que hay dos lugares en donde es posible realizar la declaración de una variable: fuera de todo bloque, entendido por bloque un conjunto de sentencias encerradas entre el carácter '{'y el carácter '}', y dentro de un bloque de sentencias.

En nuestro ejemplo, se ha declarado la variable *c* antes de la función **main** (fuera de todo bloque) y las variables *i, f* y *d* dentro de la función (dentro de un bloque). Una variable declarada fuera de todo bloque se dice que es *global* porque es accesible en cualquier parte del código desde su declaración hasta el final

del fichero fuente. Por el contrario, una variable declarada dentro de un bloque, se dice que es *local* porque solo es accesible dentro de este.

Según lo expuesto, la variable *c* es global y las variables *y, f* y *d* son locales.

INICIACIÓN DE UNA VARIABLE

Las variables globales son iniciadas por omisión por el compilador: las variables numéricas con 0 y los caracteres con '/0'. También pueden ser iniciadas explícitamente, como hemos hecho en el ejemplo anterior con la variable *c*. En cambio, las variables locales no son inicializadas por el compilador, por lo tanto, depende de nosotros iniciarlas o no.

EXPRESIONES NUMÉRICAS

Una expresión es un conjunto de operandos unidos mediante operadores para especificar una operación determinada. Cuando todas las expresiones se evalúan retornan un valor. Por ejemplo:

```
t + 1

sum + c

cantidad * precio

x = 7 * sqrt(t) – x / 2
```

1.6 OPERADORES

Los operadores son funciones a las cuales se les ha asignado un simbolo que indican cómo se manipulan los datos. Se pueden clasificar en los siguientes grupos: aritméticos, relacionales, lógicos, unitarios, a nivel de bits, de asignación, operador condicional, etc.

OPERADORES ARITMÉTICOS

Los operadores aritméticos los utilizamos para realizar operaciones matemáticas y son los siguientes:

Operador	Operación
+	*Suma*. Los operandos pueden ser enteros o reales.
-	*Resta.* Los operandos pueden ser enteros o reales.
*	*Multiplicación*. Los operandos pueden ser enteros o reales
/	*División*. Los operandos pueden ser enteros o reales. Si ambos operandos son enteros el resultado es entero, en el resto de los casos el resultado es real.

%	*Módulo* o resto de una división entera. Los operandos tienen que ser enteros.

El siguiente ejemplo muestra cómo utilizar estos operadores.

```
int t =10, b = 3, n;

float x = 2.0F, y;

y = x + t; // El resultado es 12.0 de tipo float

n = t / b; // El resultado es 3 del tipo int

n = t % b; // El resultado es 1 del tipo int

y = t / b; // El resultado es 3 de tipo int. Se convierte a float
para ser asignado a y

n = x / y; // El resultado es 0.666667 de tipo float. Se convierte
a int para asignarlo a n (n = 0)
```

Cuando en una operación aritmética los operandos son de diferentes tipos, ambos son convertidos al tipo del operando de precisión más alta.

OPERADORES DE RELACIÓN

Los operadores de relación o de comparación permiten evaluar la igualdad y la magnitud. El resultado de una operación de relación es un valor booleano verdadero o falso (1 o 0). Los operadores de relación son los siguientes:

Operador	Operación
<	El primer operando *menor que* el segundo
>	El primer operando *mayor que* el segundo
<=	El primer operando *menor o igual que* el segundo
>=	El primer operando *mayor o igual que* el segundo
!=	El primer operando *distinto que* el segundo
==	El primer operando *igual que* el segundo

Los operandos tienen que ser de un tipo primitivo. Por ejemplo:

```
int r = 10, t = 0, y = 0;

y = r == t; // y = 0 (falso) porque r no es igual a t

y = r > t; // y = 1 (verdadero) porque r es mayor que t
```

```
y = r != t; // y = 1 (verdadero) porque r no es igual a t
```

Un operador de relación equivale a una pregunta relativa sobre como son dos operandos entre sí. Por ejemplo, la expresión *r == t* equivale a la pregunta *¿x* es exactamente igual a *y*? Una respuesta *si* equivale a un valor verdadero (1) y una respuesta *no* equivale a un valor falso (0).

OPERADORES LÓGICOS

El resultado de una operación lógica (AND, OR, XOR y NOT) es un valor booleano verdadero o falso (1 o 0). Las expresiones que dan como resultado valores booleanos (véanse los operadores de relación) pueden combinarse para formar expresiones *booleanas* utilizando los operadores lógicos indicados a continuación.

Los operandos deben ser expresiones que den un resultado verdadero o falso.

Operador	Operación
&&	*AND*. Da como resultado verdadero si al evaluar cada uno de los operandos el resultado es verdadero. Si uno de ellos es falso, el resultado es falso. Si el primer operando es falso, el segundo operando no es evaluado.
||	*OR*. El resultado es falso si al evaluar cada uno de los operandos el resultado es falso. Si uno de ellos es verdadero, el resultado es verdadero. Si el primer operando es verdadero, el segundo operando no es evaluado.
!	*NOT*. El resultado de aplicar este operador es falso si al evaluar su operando el resultado es verdadero, y verdadero en caso contrario.
^	*XOR*. Da como resultado verdadero si al evaluar cada uno de los operandos el resultado de uno es verdadero y el del otro falso; en otro caso el resultado es falso.

El resultado de una operación lógica es de tipo **int** o **booleana**. Por ejemplo:

```
int o = 10, p = 0, q = 0;

q = (o != 0) && (p != 0); // q = 0 (falso)
```

Los operandos del operador *&&* son: *o != 0* y *p != 0.* El resultado de la expresión *o != 0* es verdadero porque *o* vale 10 y *p != 0* es falso porque *p* es 0. Por lo tanto, el resultado de *verdadero && falso* es falso.

OPERADORES UNARIOS

Los operadores unarios se aplican a un solo operando y son siguientes: !, - , ~, ++ y --. El operador ! ya lo hemos visto y los operadores ++ y -- los veremos más adelante.

Operador	Operación
~	Complemento a 1 (cambia los ceros por unos y unos por ceros). El carácter ~ es el ASCII 126. El operando debe de ser de un tipo primitivo entero.
-	Cambia el signo al operando (esto es, se calcula el complemento a dos que es el complemento 1 más 1). El operando puede ser de un tipo primitivo entero o real.

El siguiente ejemplo muestra cómo utilizar estos operadores:

```
int a = 2, b = 0, c = 0;

c = -a; // resultado c = -2

c = ~b; // resultado c = -1
```

OPERADORES A NIVEL DE BITS

Estos operadores permiten realizar con sus operandos las operaciones AND, OR, XOR y desplazamientos, bit por bit. Los operandos tienen que ser enteros.

Operador	Operación
&	Operación AND a nivel de bits.
|	Operación OR a nivel de bits.
^	Operación XOR a nivel de bits.
<<	Desplazamiento a la izquierda rellenando con ceros por la derecha.
>>	Desplazamiento a la derecha rellenando con el bit de signo por la izquierda.

Los operandos tienen que ser de un tipo primitivo entero.

```
int a = 255, r = 0, m = 32;

r = a & 017; //r = 15. Pone a cero todos los bits de a excepto los 4 bits de menor peso.
```

```
r = r | m // r = 47. Pone a 1 todos los bits de r que estén a 1 en m.

r = a & ~07; // r = 248. Pone a 0 los 3 bits de menor peso de a.

r = a >> 7; // r = 1. Desplazamiento de 7 bits a la derecha.

r = m << 1; // r = 64. Equivale a r = m * 2

r = a >> 7; // r = 16. Equivale a r = m / 2
```

OPERADORES DE ASIGNACIÓN

El resultado de una operación de asignación es el valor almacenado en el operando izquierdo, lógicamente después de que la asignación se ha realizado. El valor que se agina es convertido implícitamente o explícitamente al tipo del operando de la izquierda.

Operador	Operación
++	Incremento.
--	Decremento.
=	Asignación simple.
*=	Multiplicación más asignación.
/=	División más asignación.
%=	Módulo más asignación.
+=	Suma más asignación.
-=	Resta más asignación.
<<=	Desplazamiento a izquierda más asignación.
>>=	Desplazamiento a derecha más asignación.
&=	Operación AND sobre bits más asignación.
|=	Operación OR sobre bits más asignación.
^=	Operación XOR sobre bits más asignación.

A continuación se muestran algunos ejemplos con estos operandos.

```
int x = 10, y = 1;

x++; // Incrementa el valor de n en 1

x--; // Decrementa el valor de n en 1

y +=2 // Realiza la operación i = i + 2
```

1.7 SENTENCIAS DE CONTROL

En esta parte se aprenderá a escribir código para que un programa tome decisiones y sea capaz de ejecutar bloques de sentencias repetidas veces.

SENTENCIA IF

La sentencia if permite tomar una decisión para ejecutar una acción u otra, esta decisión es del tipo booleana ya sea verdadero o falso y la sintaxis es la siguiente.

```
if(condición)
     sentencia 1;
[else
     sentencia 2];
```

donde la *condición* es una expresión lógica o relacional y *sentencia 1* y *sentencia 2* representan el código que quieren que se ejecute.

Una sentencia **if** se ejecuta de la forma siguiente:

1. Se evalúa la expresión *condición.*
2. Si el resultado de la evaluación de la *condición* es verdadera se ejecutará la *sentencia 1.*
3. Si el resultado de la *condición* es falso se ejecutara la *sentencia 2.*
4. Si el resultado de la *condición* es falso y no se ha puesto la cláusula **else**, la *sentencia 1* será ignorara.

A continuación se exponen algunos ejemplos de cómo se utiliza la sentencia **if**.

```
int a = 5, b = 4;
if(a < b){
printf("a es menor que b");
}else{
printf("a no es menor que b");
}
```

En este ejemplo, la condición esta impuesta por una expresión de relación. Si al evaluar la condición se cumple que *a* es menor que *b* lo cual es falso, entonces imprimirá un mensaje el cual es *"a es menor que b"*, como sabemos que la condición es falsa se ejecuta la sentencia dos que imprime el mensaje *"a no es menor que b"*.

Con esto queda por visto la sentencia de control **if**.

Esta estructura **else if** es una estructura consecuente de la sentencia if, en la cual se evalúan diferentes casos, su forma general es:

```
if(condición 1)
      sentencia 1;
else if(condición 2)
      sentencia 2;
else if(condición 3)
      sentencia 3;
.
.
.
else
sentencia n;
```

La estructura **else if** se evalúa así: Si se cumple el primer caso (*condición 1*), se ejecuta lo que se encuentra en la *sentencia 1* y si no se cumple se examina secuencialmente los siguientes casos (*condiciones*) hasta llegar al último **else if**. Si ninguna condición es verdadera entonces se ejecutara la *sentencia n* que corresponde al último **else**. El siguiente ejemplo se muestra cómo funciona:

```
int a = 10, b = 5, c = 11;
if(a < b){
printf("a es menor que b");
}else if(a == b){
printf("a es igual que b");
}else if(a > c){
printf("a es mayor que c");
}else{
printf("Ningún caso es verdadero");
}
```

En este ejemplo podemos observar que las *condiciones* son falsas porque así lo hemos hecho, pero primero se evalúa la *condición 1* que es *a < b* y si no se cumple sigue con la *condición 2* que es *a ==b* hasta llegar a una condición verdadera que es el último **else** ya que las anteriores han sido falsas.

SENTENCIA SWITCH

La sentencia switch permite ejecutar varias acciones en función de una expresión. Es una sentencia para decisiones múltiples dado un determinado valor el cual se le da a la expresión. Su sintaxis es:

```
switch(expresión){
      case [expresión-constante 1]:
            sentencia 1;
      break;
      case [expresión-constante 2]:
            sentencia 2;
      break;
.
.
.
      default:
            sentencia n;
}
```

donde *expresión* es la variable que almacenará o recibirá los datos de cada caso (*case*). La sentencia **switch** evalúa la expresión entre paréntesis y compara su valor con las constantes de cada **case**. La ejecución de las sentencias del bloque de la sentencia **switch**, comienza en el **case** cuya constante coincida con el valor de la expresión y continúa hasta el final del bloque, si no existe ninguna variable para **case** entra al **default**, un **default** sería como un **else** poniendo como ejemplo la sentencia **if**.

Existen también otras sentencias, como lo es la sentencia **while** que se seguía ejecutando dependiendo de su *condición* y la sentencia **do while** que es muy parecida a **while** pero estas se dejan al estudio del lector.

SENTENCIA FOR

La sentencia **for** nos permite ejecutar una o varias líneas de código repetitivamente a un número determinado de veces. Su sintaxis es:

```
for(val1 = val2/const; condición 1 ; condición 2){

sentencia;

}
```

La sentencia for se evalúa de la siguiente forma:

1. A **val1** se le asigna un valor **constante** o el valor de alguna otra variable.
2. Se evalúa la condición:
    a. Si la **condición 1** es verdadera respecto a **val1,** se ejecuta la sentencia respecto a **condición 2** y así sucesivamente dependiendo de esta.
    b. Si la **condición 1** es falsa respecto a **val1,** la sentencia **for** termina.

Ahora veremos algunos ejemplos, el primer ejemplo nos imprimirá los valores desde el 0 al 100, existen dos modos de hacer esto y depende de la **condición 1**.

```
for(int i = 0; i <= 100 ; i++){

printf("%d", i);

}
```

Como podemos ver en este ejemplo la variable **i** comienza en el valor constante **0**, la primera condición declara que tiene que ser menor o igual a **100**, o sea que llegará al valor a **100**, ahora en la **condición 2** que ya se ve vio anteriormente hace un incremento en **1**. Ahora en el siguiente ejemplo la condición 1 cambia pero hace lo mismo que el primer ejemplo.

```
for(int i = 0; i < 101 ; i++){

printf("%d", i);

}
```

La explicación es igual a la anterior solo que en la **condición 1** se evalúa un **<** explicito, o sea que imprimirá hasta el número **100**, pero cuando llegue a **101** se detiene la sentencia for.

```
for(int i = 100; i >= 1 ; i--){

printf("%d", i);

}
```

Este ejemplo es similar a los anteriores pero al revés, empieza en el número **100** y se hace un decremento en 1 cuando se evalúa la **condición 1**, como en las explicaciones anteriores.

Video [tutorial]: *21. Programación en C++ || Ciclos o Bucles || La sentencia for*

CAPÍTULO 2

SOFTWARE PARA EL FUNCIONAMIENTO DE NUESTRA TARJETA

2.0 DESCARGA E INSTALACIÓN DEL SOFTWARE

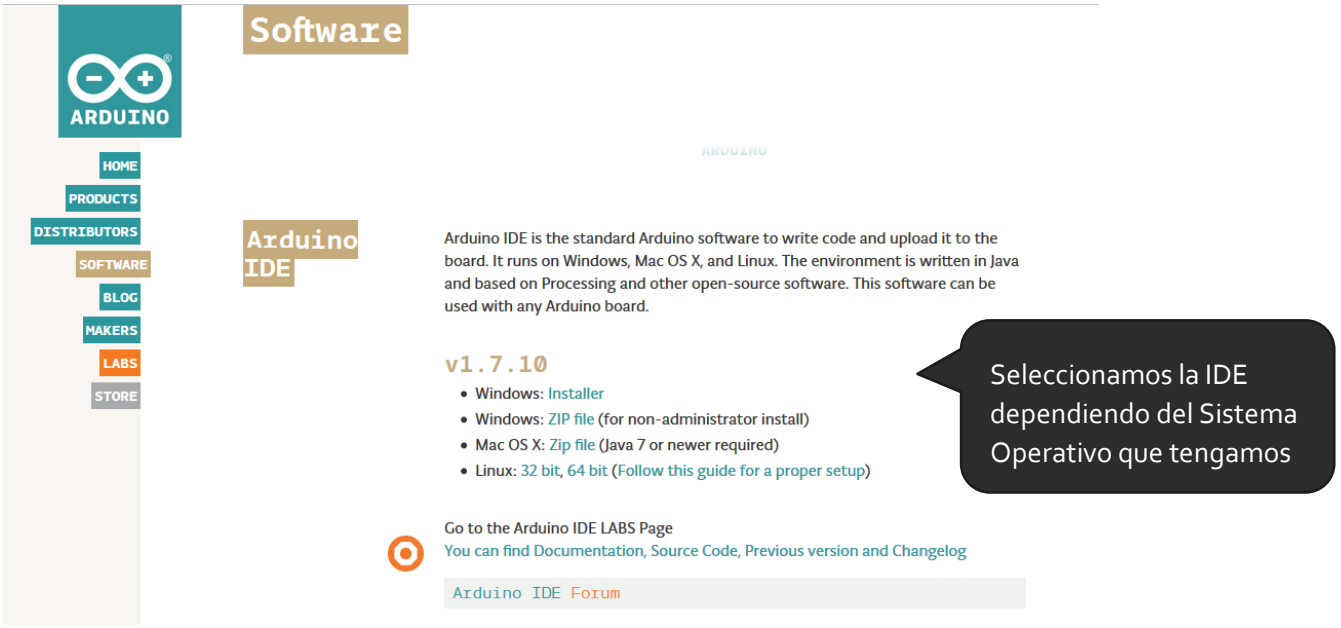
En este capítulo se aprenderá a descargar el software necesario para editar y programar nuestra tarjeta de desarrollo Arduino, se seguirá paso a paso sobre descargarlo de internet e instalarlo en nuestra computadora.

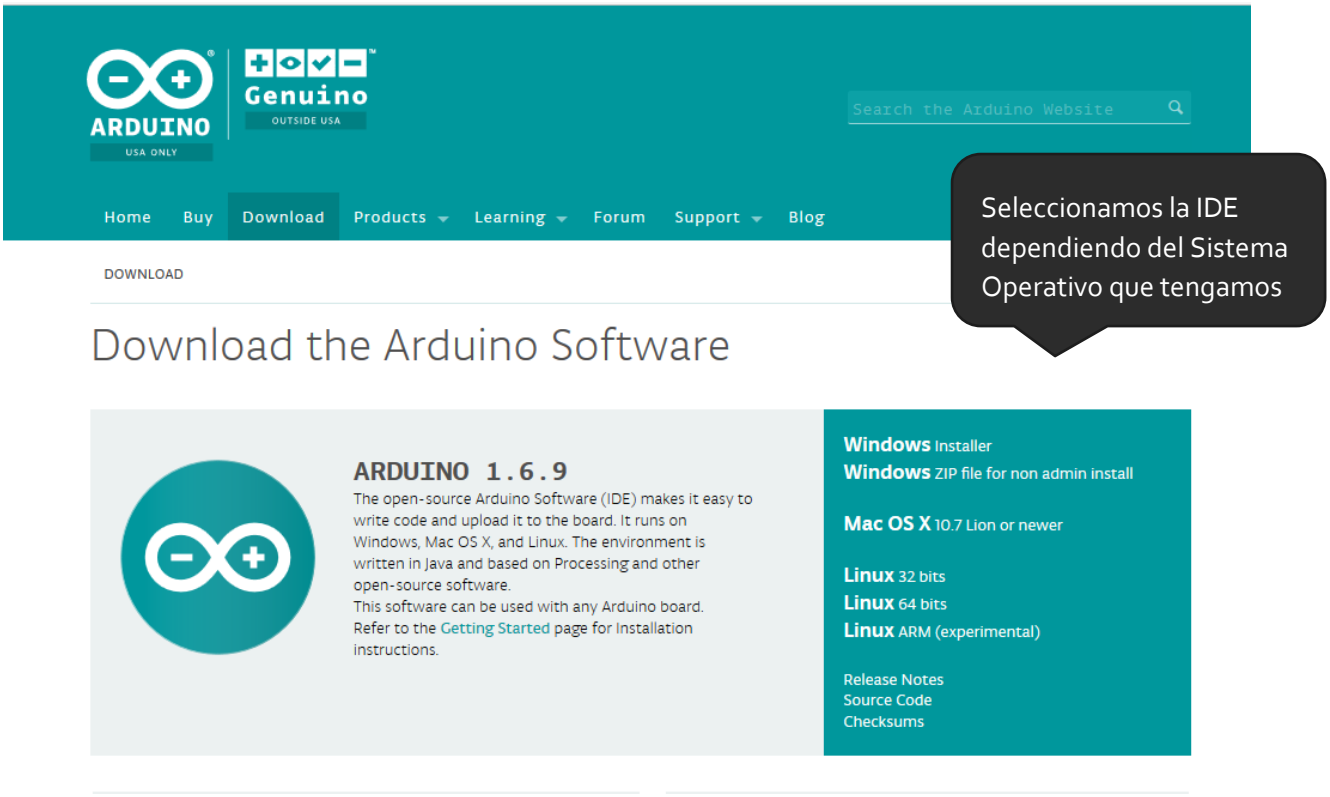
2.1 DESCARGA EL IDE ARDUINO

Este programa lo descargaremos de Internet, tenemos dos opciones aunque ambas son exactamente lo mismo son versiones diferentes, una es de Arduino.org y la otra de Arduino.cc puede escoger la que guste y luego realizar los siguientes pasos:

- Una vez en la página ya sea la de .org o .cc iremos a **software** o **download**, como ilustra la siguiente imagen:

- Aparecerá una nueva ventana en la cual usando el scroll nos deslizaremos hacia abajo y seleccionaremos el tipo de IDE a emplear ya sea que tengamos Windows, Linux o MAC OS





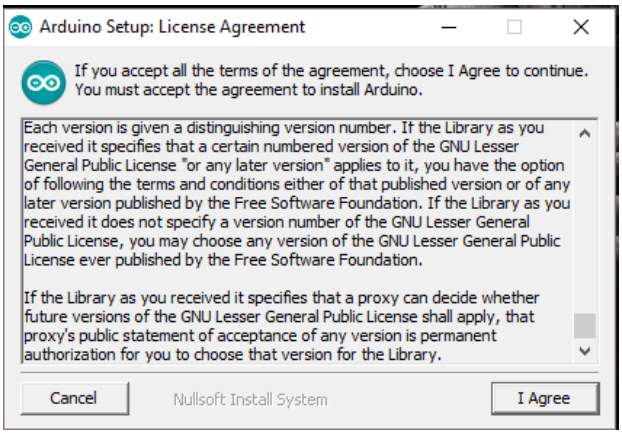
- Inmediatamente saldrá la pantalla para descargar el archivo, este tendrá el nombre de **arduino-…** y daremos clic en **guardar archivo.**

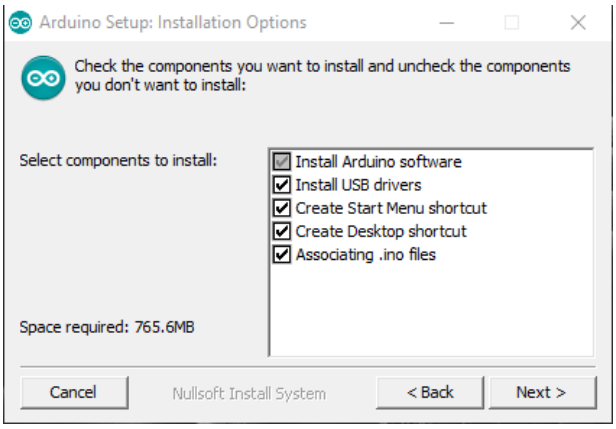
Le daremos clic en Guardar archivo y comenzara la descarga y se guardad en nuestra carpeta de Descargas en nuestros Documentos

Share

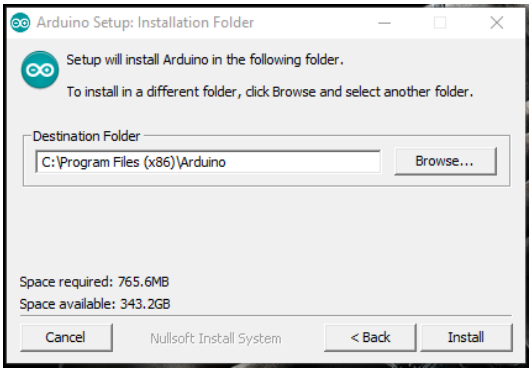
2.2 INSTALACIÓN DEL IDE

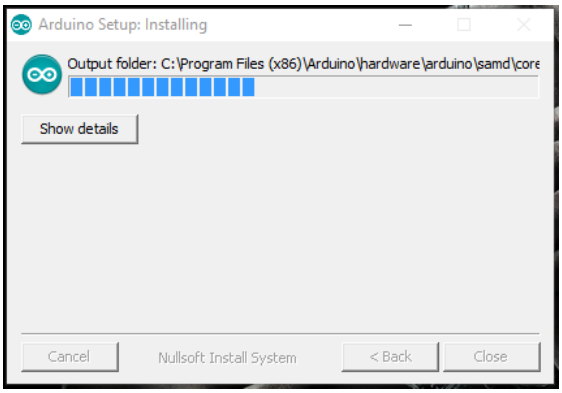
Para instalar el Entorno de Desarrollo Integrado (IDE), se debe ejecutar el archivo previamente descargado que se encuentra en nuestra carpeta de **Descargas** y ejecutar el archivo **arduino-**.exe,** aparecerá una ventana la cual nos indica los términos y condiciones del programa, nosotros daremos clic en **I Agree** y después aparecerá otra ventana y daremos clic en **Next**.

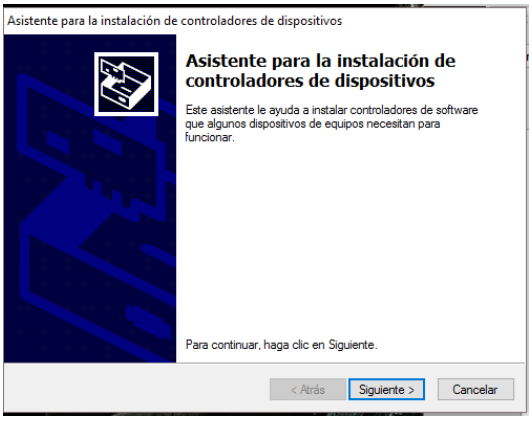


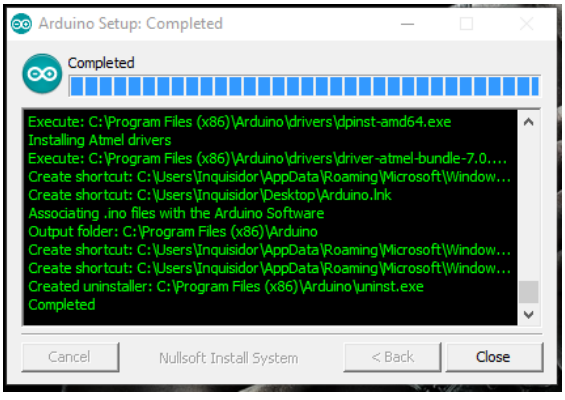


Aparecerá una nueva ventana en donde se muestra el lugar donde se va a instalar el IDE, nosotros no modificaremos nada y la dejaremos como está, daremos clic en **Install** y empezará la instalación, esperaremos unos minutos a que termine la instalación, nos aparecerá la ventana de los controladores le daremos en **Siguiente** y a lo que nos pida instalar y finalmente cuando se termine de instalar presionamos **Close** y tendremos nuestra IDE instalada.









CAPÍTULO 3

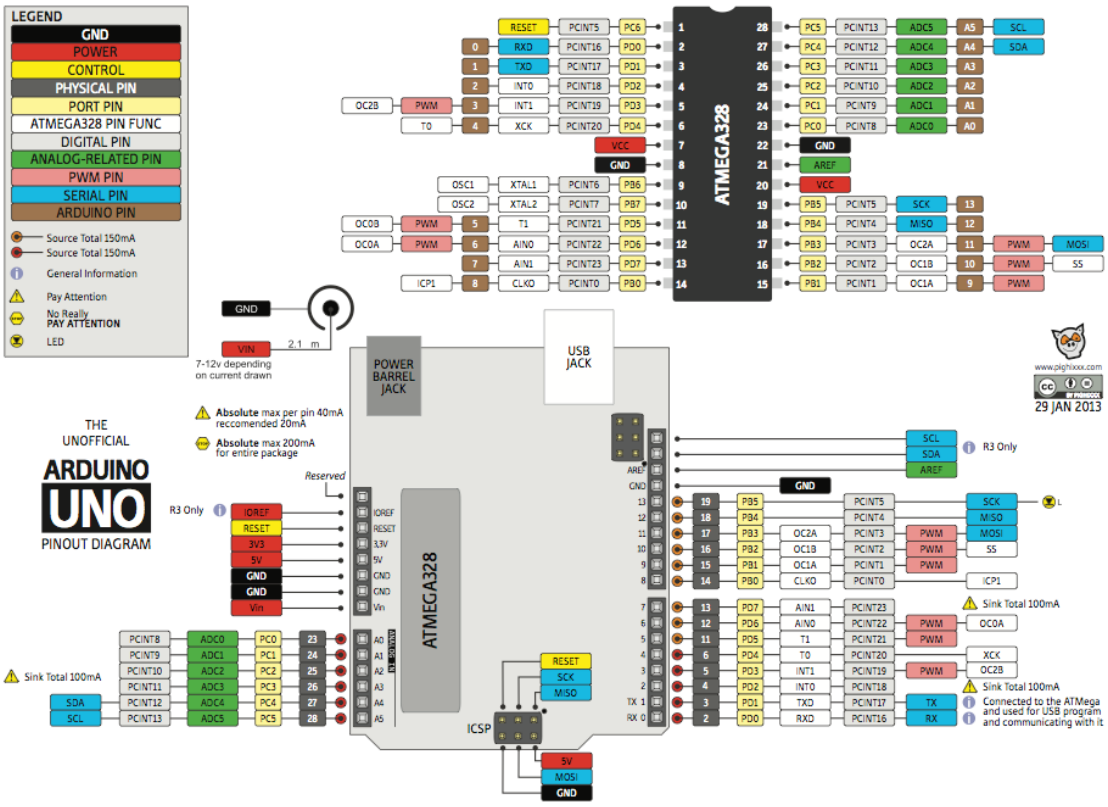
INTRODUCCIÓN AL ARDUINO

3.0 ¿QUÉ ES ARDUINO?

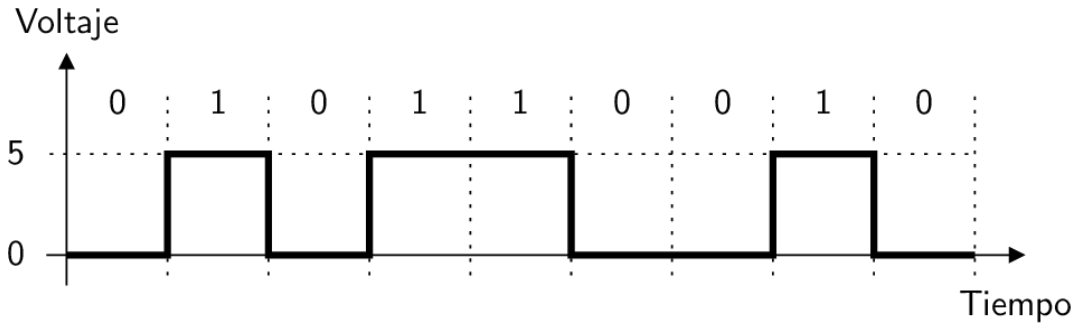
Las tarjetas de desarrollo Arduino son una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se ha creado para artistas, diseñadores, aficionados y cualquier interesado en crear entornos u objetos interactivos, debemos recordar que Arduino es Open Source y cualquiera puede contribuir al desarrollo de esta plataforma con nuevas ideas o proyectos.

3.1 ESTRUCTURA DE ARDUINO

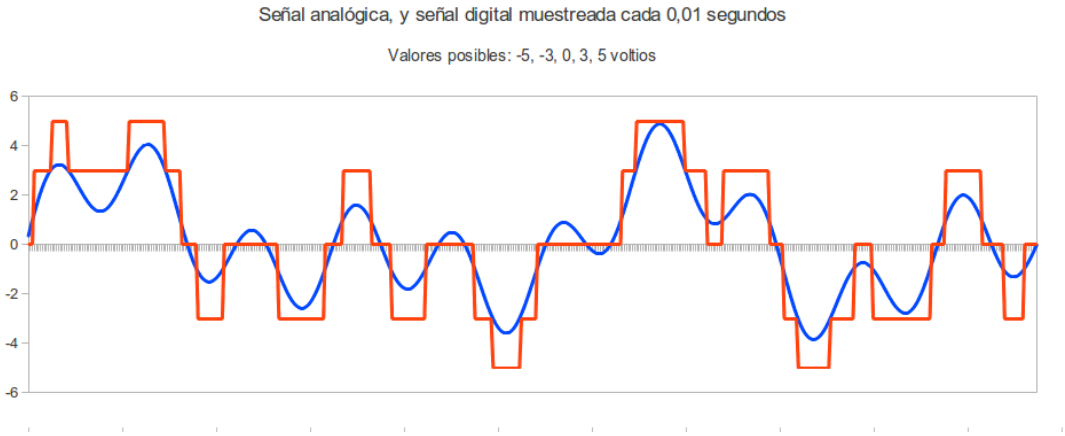
Prácticamente veremos la tarjeta Arduino UNO pero en general las diferentes tarjetas de Arduino son muy similares, cuentan con pines tanto de salida como de entrada con los cuales podremos leer nuestros dispositivos ya sea una señal de algún sensor u otro parámetro. También enviar señales o datos por los pines de salida los cuales veremos cómo funcionan más adelante para usar los Actuadores analógicos y digitales. Aquí se presenta una imagen de la estructura, recordemos que usa un microcontrolador ATMEGA328 para que funcionen todos nuestros dispositivos.



Nuestra tarjeta tiene 14 pines digitales del 0 al 13, de los cuales podemos leer y enviar señales digitales que van de 0 a 5 volts, además entre esos pines se cuenta con 6 pines PWM los cuales los veremos más adelante, aquí una imagen de señal digital:



También cuenta con 6 pines analógicos, como lo dice su nombre podremos hacer lecturas analógicas igualmente de 0 a 5 volts, estos pines también se pueden usar como digitales por medio del **convertidor analógico digital**, aquí una imagen de señal analógica:



Recordemos que cada pin trabaja con voltajes de 0 a 5 volts CC, además que la máxima corriente por pin es de 40mA, si utilizaremos un actuador que pide más corriente que la entregada por un pin es necesario usar un transistor de potencia, pero eso se verá más adelante.

La tarjeta consta de un regulador de voltaje, un 7805 conectado al **Jack** y al pin **vin** de la tarjeta con su respectivo diodo de protección, pero esto no evita el tener precaución y no invertir la polaridad en los pines de alimentación haciendo un cortocircuito.

Nota: Nunca alimentar la placa Arduino por el puerto USB ya que los puertos USB de las computadoras entregan 5 volts y esta señal no va al regulador, la tarjeta no fue creada para alimentarla de esta forma al querer ejecutar un proyecto.

Video [tutorial]: *[Curso Arduino Básico - #1] Introducción al Arduino*

Esta tabla podremos ver las funciones de cada pin:

PIN	NOMBRE	DESCRIPCION
2	PIN 0/RX	Pin I/O, entrada de dato, comunicación serial.
3	PIN 1/TX	Pin I/O, salida de dato, comunicación serial.
4	PIN 2/INT0	Pin I/O, resistencia pull up, interrupción ext.
5	PIN 3/INT1/PWM	Pin I/O, resistencia pull up, interrupción ext.
6	PIN 4/T0	Pin I/O, resistencia pull up, entrada reloj TIMER0.
11	PIN 5/T1/PWM	Pin I/O, resistencia pull up, salida reloj TIMER0, PWM salida o entrada.
12	PIN 6/AIN0/PWM	Pin I/O, resistencia pull up, salida reloj TIMER0, comparador, PWM salida o entrada.
13	PIN 7/AIN1	Pin I/O, resistencia pull up, comparador.
14	PIN 8/CLKO	Pin I/O, resistencia pull up, salida ¼ frecuencia osc.
15	PIN 9/OC1A/PWM	Pin I/O, resistencia pull up, salida reloj TIMER1, PWM salida o entrada.
16	PIN 10/OC1N/PWM	Pin I/O, resistencia pull up, salida reloj TIMER1, PWM salida o entrada.
17	PIN 11/OC2A/PWM	Pin I/O, resistencia pull up salida reloj TIMER2, PWM salida o entrada.
18	PIN 12	Pin I/O, resistencia pull up, MISO.
19	PIN 13/LED	Pin I/O, resistencia física, LED.
23	A0/ADC0	Pin I/O, entrada comparador analógico.
24	A1/ADC1	Pin I/O, entrada comparador analógico.
25	A2/ADC2	Pin I/O, entrada comparador analógico.
26	A3/ADC3	Pin I/O, entrada comparador analógico.
27	A4/ADC4/SDA	Pin I/O, entrada comparador analógico, interfaz i^2c.
28	A5/ADC5/SCL	Pin I/O, entrada comparador analógico, interfaz i^2c.

3.2 CONCEPTOS BÁSICOS

Es importante seguir estas recomendaciones, ya que si no lo hace podría dañar su tarjeta de desarrollo:

1. No sobrepase los niveles de corriente, tanto de entrada como de salida, recuerde que las tarjetas Arduino entregan un máximo de **40mA** por pin. Así mismo también soporta una corriente máxima de 40mA de lectura, esto quiere decir que puede encender un led con una resistencia de **220 Ω**.

El voltaje de salida de los pines de la tarjeta Arduino es de 5V y la corriente que requiere un led para encenderlo es de **20mA** así que, veamos qué resistencia necesitamos para encenderlo correctamente:

$$V = R \times I \qquad R = \frac{V}{I} \qquad R = \frac{5V}{0.020A} \qquad R = 250\Omega \approx 220\Omega{\sim}270\Omega$$

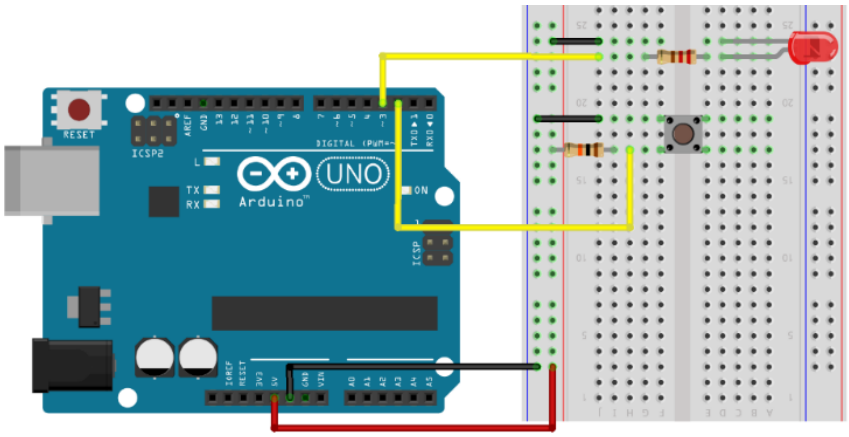
Ahora veremos un ejemplo de la corriente de entrada en nuestra tarjeta, supongamos que queremos usar un pulsador, veremos qué resistencia debemos colocarle la cual será la mínima, como ya sabemos la corriente máxima de entrada es de **40mA**:

$$V = R \times I \qquad R = \frac{V}{I} \qquad R = \frac{5V}{0.040A} \qquad R = 125\Omega \approx 120\Omega{\sim}150\Omega$$

Como hemos mencionado es el mismo valor que debe colocarse para estar al límite de la capacidad soportada por el la Arduino, pero esto no es aconsejable por lo que, en la mayoría de los casos se usa una resistencia de **1KΩ** a **10KΩ**, así nuestra tarjeta trabaja perfecta y tranquilamente con **5mA**.

2. En proyectos externos se recomienda usar una fuente de voltaje de 12V y ser alimentado por el **Jack** de nuestra tarjeta y no por el puerto **USB**, como ya se ha mencionado antes.
3. Tratar de que mientras nuestra tarjeta esté en funcionamiento que no caiga ningún líquido en ella o alguna pieza de metal que haga contacto entre si, ya que podría sufrir algún corto circuito y estropear la misma.
4. Si es posible adquirir una carcasa de acrílico para mantenerla completamente aislada y así no sufra daño alguno.
5. Si usa sensores, relevadores, servomotores, etc., emplear usar una fuente externa para alimentar los mismos, se recuerda que cada pin de nuestra tarjeta entrega un máximo de **40mA**, si usted sobrecarga la corriente quemará su tarjeta.

Con estas recomendaciones usted ya podrá trabajar tranquilamente con su tarjeta y crear los proyectos que se presentan están a continuación o los que usted quiera, en seguida se presenta el diagrama del punto 1:



3.1 EL IDE DE ARDUINO

Ahora veremos brevemente cómo emplear el IDE de Arduino, ya que es muy simple de usar, el IDE consta de una barra de herramientas que son:

- Archivo.
- Editar.
- Programa.
- Herramientas.
- Ayuda.

En **Archivo** tenemos las clásicas opciones de cualquier editor, como guardar, nuevo archivo, **ejemplos**, en esta opción vienen ejemplos de programación que Arduino ha puesto, también tiene opciones como imprimir y preferencias.

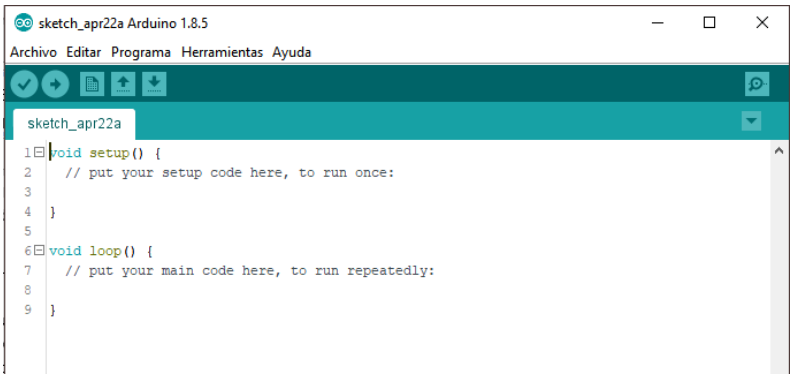
En **Editar** podemos configurar la letra tipo sangría, el modo de texto, formato, buscar, comentar.

En **Programa** tenemos las opciones de compilar, subir, incluir librería, subir con programador. En esta sección la que más nos interesa es la de incluir librería.

En **Herramientas** tenemos las opciones de autoformato, monitor serie, serial plotter, placa, procesador, puerto, programador y otras más, las que nos interesan son **placa**, **procesador** y **puerto**, ya que en la opción **placa** seleccionaremos el tipo de Arduino que tenemos (el que estamos conectando), en la opción **puerto** seleccionaremos el puerto donde está conectada nuestra placa COM1, COM2, etc., en **procesador** seleccionaremos qué tipo de microcontrolador tiene nuestra placa, o sea el modelo o versión.

Ya dentro del editor tenemos **setup()** y **loop()**, en setup configuraremos la inicialización de los pines y todo lo que queramos que se inicie por primera vez, y en loop estará en nuestro programa de ejecución, además arriba de setup irán nuestras variables globales, los define, y las librerías a usar y debajo de loop nuestros métodos o funciones.

Aquí tenemos una imagen del IDE:



Video [tutorial]: *[Curso Arduino Básico - #2] Introducción al Arduino*

CAPÍTULO 4

PROYECTOS CON ARDUINO

Nota: El código de programación estará en color rojo y los comentarios en azul.

4.0 ACTUADORES DIGITALES

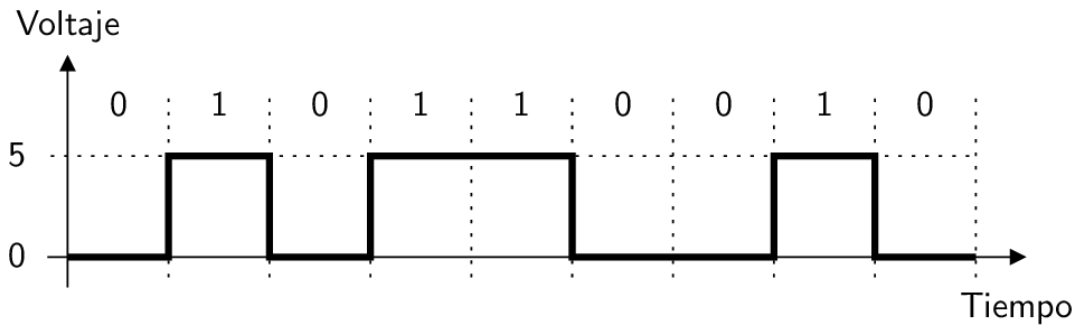
Primero explicaremos qué es una señal digital y como podremos trabajar con una en nuestra tarjeta Arduino.

¿Qué es algo digital?

- Digital.
    - Solo dos valores.

Ejemplos:

- [0-1].
- Verdadero o Falso.
- 0v o 5v.



¿Qué quiere decir esto? Que solo tendremos pulsos de 0v y 5v por un determinado tiempo 1s, 1/2s, etc.

Pines digitales de la tarjeta Arduino UNO.

El Arduino UNO tiene 14 pines digitales, cada uno de estos para:

- Entrada y salida digital.
- Voltajes de trabajo de 0v o 5v.
- Corriente máxima por pin de 40mA.

Se debe de tener en consideración que Arduino tomará como un pulso **alto** o **1** a partir de 3v y como un pulso **bajo** o **0** a partir de 2v (valores aproximados).

Configuración de pines digitales.

Esta parte es muy importante ya que veremos las "palabras reservadas" del lenguaje de programación de Arduino, estas nos servirán para poder poner un pin ya sea como entrada o salida, en este caso se ven los actuadores digitales así que los pines serán usados solo como salida, y esto es esencial ya que uniendo esto con los siguientes temas veremos que, podemos hacer muchas cosas interesantes.

- Los pines pueden usarse como entrada o salida.
- Para definir su funcionamiento, en entrada o salida emplearemos la función **pinMode():**
    - Ejemplo:
        - pinMode([PIN/ALIAS],[INPUT/OUTPUT]);
        - Donde INPUT – es de enterada y OUTPUT – es de salida.

Como bien ya se ha mencionada varias veces y no está de más decirlo de nuevo, la salida será de 0v o 5v, aquí tenemos un ejemplo un poco más concreto.

- Los pines los configuramos dentro del setup(){}
    - Ejemplo:

```
setup(){
pinMode(2,OUTPUT);
}
```

- Para definir o escribir una salida usamos la función digitalWrite, esta función enviará un pulso alto o bajo:
    - Ejemplo:
        - digitalWrite([PIN],[HIGH/LOW]);
        - Donde HIGH será 5v y LOW será 0v

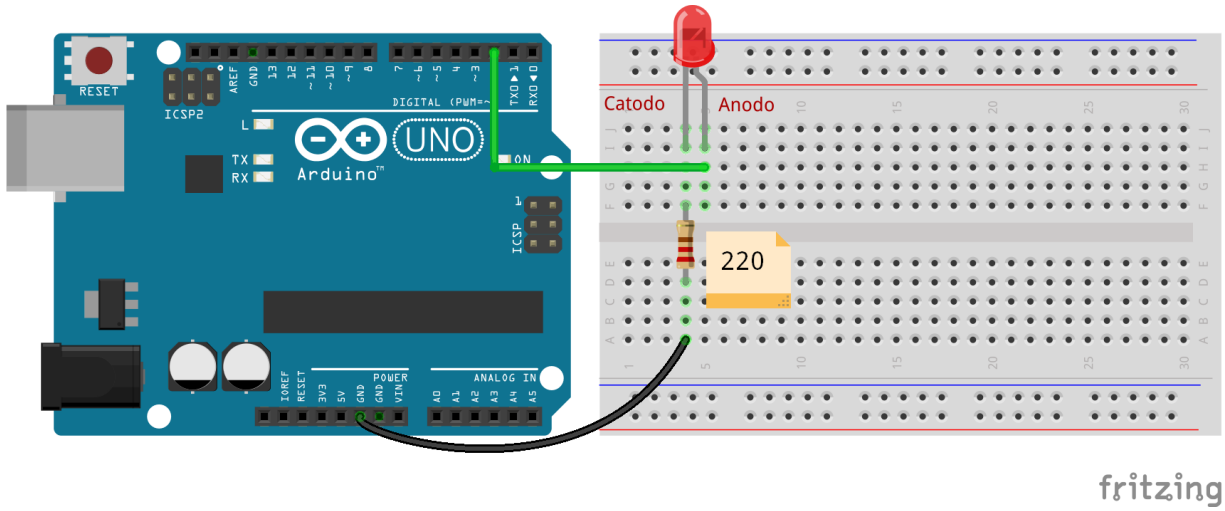
4.0.1 MI PRIMER PROGRAMA EN ARDUINO "BLINK"

Comenzaremos haciendo el programa clásico del blink que en otro lenguaje de programación sería como el "Hola Mundo" ya que es el primer programa que se enseña, el código será el siguiente. Es muy importante que usted escriba el código para que pueda aprender correctamente y se vaya acostumbrando a la programación:

```
#define LED 2 //Damos el alias a nuestro pin 2.
void setup(){
pinMode(LED,OUTPUT); //Definimos el pin LED como salida.
}
void loop(){ //Definimos nuestra secuencia.
digitalWrite(LED,HIGH); //Mandamos un ALTO al LED.
delay(1000); //Tiempo en que permanece el LED prendido "un segundo".
digitalWrite(LED,LOW); //Mandamos un BAJO al LED.
delay(1000); //Tiempo en que permanece el LED apagado "un segundo".
}
```

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
Protoboard
Jumpers

Circuito:



Explicación:

Recordando lo que se vio en capítulos pasados usamos la palabra reservada **#define** para darle un alias a nuestro pin 2, el cual es **LED**, esto para tener un código mucho más limpio e identificar lo que conectamos en cada pin de nuestro Arduino, por eso **#define LED 2.** Todo esto se hace arriba del **void setup()**, también en esta parte se definen las variables para almacenar datos, o sea **variables globales**.

Ahora dentro de **void setup()** configuramos nuestro pin como salida ya que enviaremos señales digitales de **ALTO** o **BAJO**, esto se hace mediante la función **pinMode()** que en su traducción al español sería "**pin en modo de"** , esta función recibe dos parámetros, uno que es el **ALIAS** o **PIN** y la otra parte que sería la palabra reservada **INPUT** u **OUTPUT**, en este caso usamos la segunda que es **OUTPUT**, ya que enviaremos pulsos y por eso lo configuramos como salida.

Lo siguiente será configurar nuestro **programa de ejecución**, esto entra en **void loop()**, lo que escribamos dentro de él se ejecutara infinitamente hasta que nosotros cortemos el ciclo por algún otro medio. Como queremos hacer el clásico blink, lo que quiere decir hacer prender y apagar un led por un determinado tiempo, tenemos que enviar un pulso **ALTO**, para este caso usaremos la función **digitalWrite()** que traducida al español será **"escribe en digital un"**, como pueden observar también recibe dos parámetros, uno es el **ALIAS** o **PIN** y la otra parte lo que queremos **escribir,** un **ALTO** o un **BAJO** y esto se hace mediante **HIGH** o **LOW**, en nuestro caso como queremos que prenda el led enviará un alto por un determinado

tiempo, así que usamos la función **delay()**. Esta función recibe un parámetro ya sea una variable o un numero entero, este valor representa a los milisegundos que quiere decir esto que un segundo serán mil milisegundos, por eso **delay(1000)**, este será el tiempo en que el led permanezca prendido. Ahora queremos que se apague, usamos de nuevo la función **digitalWrite()**, pero ahora enviamos un pulso bajo que será **LOW** y de nuevo queremos que esto suceda solo por un segundo así que volvemos a usar **delay(1000)**, esto hará que nuestro led permanezca apagado durante un segundo, una vez que se haya echo la programación y el circuito finalmente se cargara el programa a nuestra tarjeta Arduino y veremos cómo se ejecuta nuestro blink infinitamente.

Muy bien, ya has hecho tu primer programa y circuito en Arduino, sigue así.

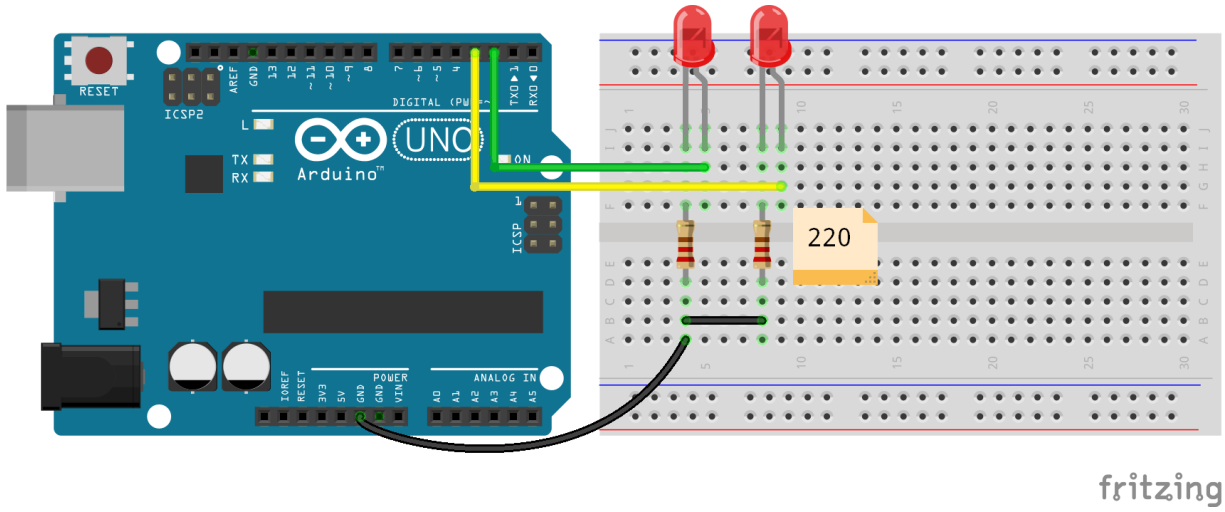
4.0.2 PARPADEO DE DOS LEDS

Ahora lo que realizaremos en esta segunda práctica es hacer parpadear dos leds simultáneamente, la explicación es similar a la anterior solo que ahora son dos leds, se recomienda que ustedes aparte de esto aumente más leds, tres, cuatro, etc., y jueguen con los tiempos del **delay()**, el código será:

```
#define LED 2 //Damos el alias a nuestro pin 2.
#define LED2 3 //Damos el alias a nuestro pin 3.

void setup(){
pinMode(LED,OUTPUT); //Definimos el pin LED como salida.
pinMode(LED2,OUTPUT); //Definimos el pin LED2 como salida.
}
void loop(){ //Definimos nuestra secuencia.
digitalWrite(LED,HIGH); //Mandamos un ALTO al LED.
digitalWrite(LED2,HIGH); //Mandamos un ALTO al LED2.
delay(1000); //Tiempo en que permanece el LED prendido "un segundo".
digitalWrite(LED,LOW); //Mandamos un BAJO al LED.
digitalWrite(LED2,LOW); //Mandamos un BAJO al LED2.
delay(1000); //Tiempo en que permanece el LED apagado "un segundo".
}
```

MATERIALES
1 ARDUINO
2 LED de 5mm
2 Resistencia de 220Ω
Protoboard
Jumpers

Circuito:



4.0.3 SECUENCIA DE LEDS

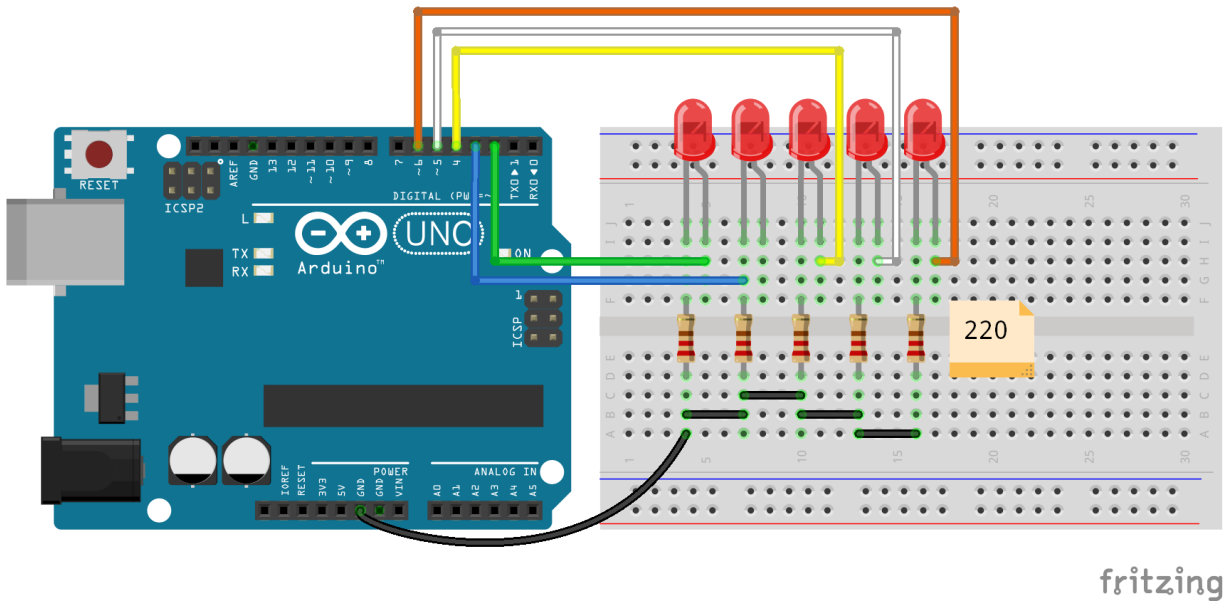
En esta tercera práctica se harán las famosas **luces del auto fantástico**, este nombre lo obtienen de la serie Knight Rider, ya que el "ayudante" del protagonista era un automóvil con inteligencia artificial que podía hablar y su boca era una serie de luces que se movían de izquierda a derecha, y es justo lo que haremos, por eso el nombre que se le da a la práctica, abajo se muestra el código, material y explicación de la misma.

MATERIALES
1 ARDUINO
5 LED de 5mm
5 Resistencia de 220Ω
Protoboard
Jumpers

```
/**
 * Programa que muestra la secuencia de leds
 */
int pausa = 50;

void setup() {
  // Inicializamos los pines del 2 al 6 como OUTPUT con un ciclo for
  for (int pinLed = 2; pinLed < 7; pinLed++)  {
    pinMode(pinLed, OUTPUT);
  }
}
```

```
void loop() {
  // Encendemos y apagamos en un loop desde el pin 2 hasta el pin 6
  for (int pinLed = 2; pinLed < 7; pinLed++) {
    // Pone el pinLed en HIGH encendiendo el led
    digitalWrite(pinLed, HIGH);
    delay(pausa);
    // Pone el pinLed en LOW apagando el led
    digitalWrite(pinLed, LOW);
  }

  delay(pausa);

  // Encendemos y apagamos en otro loop desde el pin 7 hasta el pin 2
  for (int pinLed = 6; pinLed > 1; pinLed--)   {
    // Pone el pinLed en HIGH encendiendo el led
    digitalWrite(pinLed, HIGH);
    delay(pausa);
    // Pone el pinLed en LOW apagando el led
    digitalWrite(pinLed, LOW);
  }

  delay(pausa);

}
```

Circuito:



(Hay un error en este diagrama el cable azul debe estar en el ánodo)

Explicación:

Como habrán notado tenemos unas cuantas modificaciones muy interesantes, primero creamos una variable global del tipo entero llamada **pausa** con un valor de 50, esta variable nos servirá para cambiar el tiempo en que cambian de estado los leds, si disminuimos el valor aumenta la velocidad cuando prenden y se apagan, o sea se "mueven" más rápido, si aumentan el valor es más lento.

Lo siguiente es configurar los pines como salida y aquí vemos lo interesante, no usamos la palabra reservada **define** para darle un alias a cada pin de Arduino, donde conectamos los leds, los usaremos como números y los configuraremos de "golpe" como salidas, esto lo hacemos con un **for** que ya se vio su funcionamiento en un capítulo anterior, este inicia en el pin 2 y en la **condición 1** ponemos que nuestra variable sea menor que 7 ya que será de 2 a 6 los pines a utilizar, y en la **condición 2** hacemos el incremento en 1, dentro del for usamos el **pinMode()** y le pasamos la variable del for, la ponemos como **OUTPUT** y es así como configuramos los pines como salida de "golpe" .

Ahora en el **loop** ponemos nuestro programa de ejecución, de nuevo usamos un for como el del **setup()**, y adentro del **for** ponemos la sentencia que hace que prenda el led con **digitalWrite()**, a esta función le pasamos como primer parámetro la variable del for y enviamos un alto, el led prenderá durante el tiempo de la variable **pausa** y después de ese tiempo se apagará ese mismo pin, así sucesivamente, después se le da un tiempo con un **delay()**, de nuevo le pasamos como parámetro la variable **pausa** y esto servirá para darle un tiempo para hacer el cambio al segundo **for**.

En este segundo **for** hacemos lo mismo pero ahora al revés, empezamos en el led conectado en el pin 6 y así seguimos hasta el led 2, para que esto funcione en la **condición 1** usamos un **>** y en la **condición 2** se hace un decremento, esto ya se vio en un ejemplo de un capítulo anterior.

Muy bien, ya pasaste el primer capítulo sobre actuadores digitales, ya puedes manejar cualquier actuador digital que reciba señales de 0 o 5v.

Video [tutorial]: *[Curso Arduino Básico - #3] Actuadores Digitales*

4.1 SENSORES DIGITALES

Bien, ya vimos actuadores digitales, aprendimos como escribir/enviar señales digitales de 0 o 5v. Ahora veamos como leer señales de 0 o 5v con nuestro Arduino.

- Los sensores digitales pueden estar abiertos o cerrados, o sea pueden tener el valor de 0 o 5v.
- La entrada de lectura será de 5 o 0 volts.
    - Esto sería: 1 o 0 lógicos.
- Para saber su estado utilizaremos la función:
    - digitalRead([numpin/alias]);
    - Nos devolverá un valor lógico/booleano.
        - HIGH/1 – Si está cerrado = 5 volts o mayor a 3 volts.
        - LOW/0 – Si está abierto = 0 volts o menor a 2 volts.

La lectura que se hace con **digitalRead()** y ésta se tiene que almacenar en una variable del tipo natural, o ósea byte o int, para poder usar ese valor, usando las sentencias if, switch, etc.

Tenemos algunas cuestiones antes de comenzar para que no haya algún error cuando se hace la lectura de cualquier sensor digital:

- Si el sensor no está bien conectado:
  - Tendremos un pun pin "flotante" con valores de lectura aleatorios entre 0 y 1.
- Es necesario conectarlo bien:
  - A tierra, con una resistencia pull-down.
  - A 5 volts, con una resistencia pull-up.
  - Leer la hoja de datos del sensor para ver las especificaciones.
- El pin 13 no es recomendable usarlo en la tarjeta para lectura:
  - Tiene una resistencia propia física.

Ahora veremos qué es una resistencia pull-down y pull-up y cómo se usará.

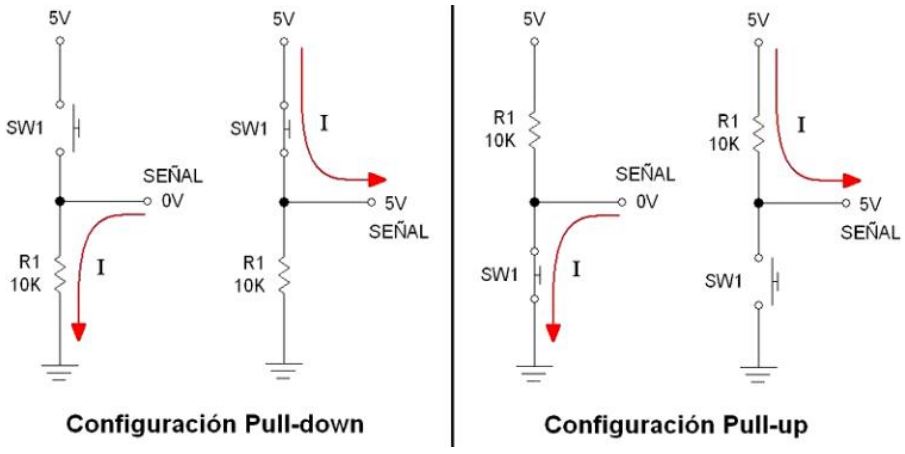
Las resistencias pull-up o pull-down se usan para establecer el estado lógico de la lectura que haremos de nuestro sensor y ésta sea correcta, como es el caso si usamos pulsadores, dip switch, sensores IR, etc.

Como bien lo dice su nombre, dependiendo de la configuración que hagamos tendremos un estado **alto** por default o un estado **bajo** también por default, cuando nuestro sensor esté en reposo y cuando activemos el sensor cambiará el estado de éste y se usan resistencias de 1k a 10k, y esto se explicó anteriormente.

En la configuración de resistencia **pull-up** en el estado de reposo tendremos un estado de **5V (HIGH)**, en cambio cuando activamos nuestro sensor se deriva toda la corriente a tierra y la caída de tención es de **0v (LOW)**.

En la configuración de resistencia **pull-down** en el estado de reposo tendremos un estado de **0V (LOW)**, en cambio cuando activamos nuestro sensor dejará pasar la corriente y tendremos una diferencia de potencial de **5v (LOW)**.

Aquí mostramos la conexión de cada una:



Esta configuración sirve solo para sensores donde solo tenemos dos estados **ALTO o BAJO**.

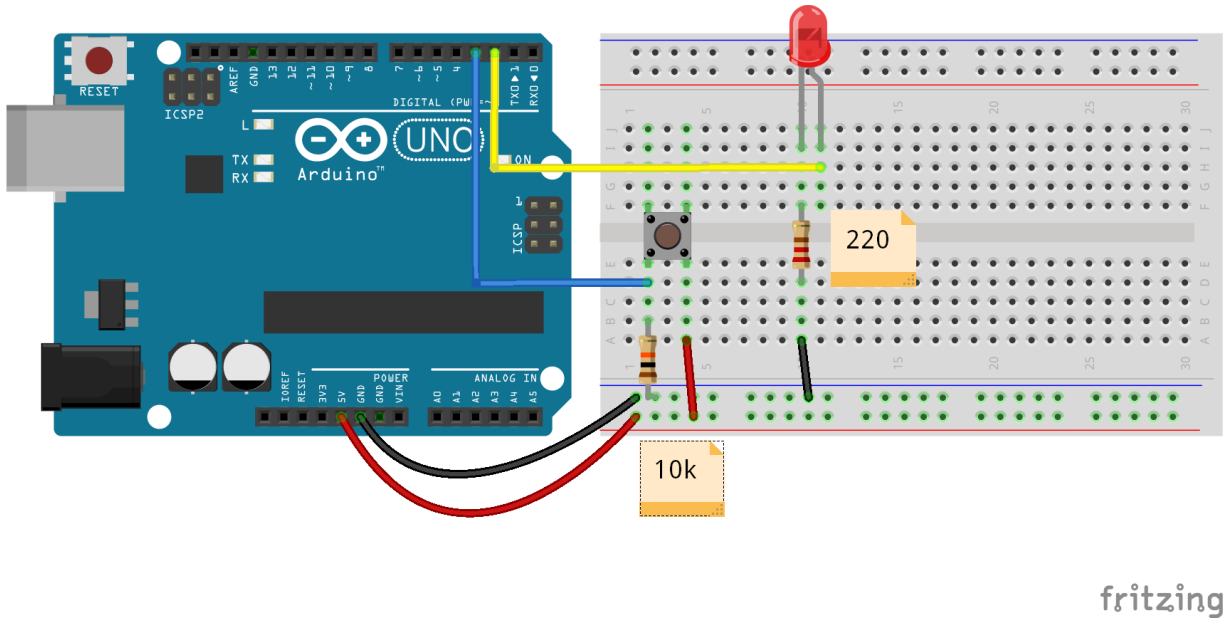
4.1.1 SENSANDO UN PULSADOR

En nuestra primera práctica vamos a leer un pulsador y por medio de una condición vamos a prender y apagar un led, esta primera práctica será la base para poder leer cualquier otro sensor con solo dos estados que son **ALTO o BAJO** y poder utilizarlo dependiendo de nuestras necesidades, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
1 Pulsador de 4 pines o 2 pines
1 Resistencia de 10k
Protoboard
Jumpers

```
#define LED 2 //Damos el alias a nuestro pin 2.
#define Push 3 //Damos el alias a nuestro pin 3.

byte Push_lee = 0;

void setup(){
pinMode(LED,OUTPUT); //Definimos el pin LED como salida.
pinMode(Push,INPUT); //Definimos el pin Push como entrada.
}

void loop(){
//Definimos nuestra secuencia.
Push_lee = digitalRead(Push); //Almacenamos en la variable la lectura
del pulsador.
 if(Push_lee == 1){ //Condición que si tenemos el valor de 1 entra en
el.
  digitalWrite(LED,HIGH); //Se manda un ALTO al LED
 }else{ //Condición que se cumple si no se cumple el if
  digitalWrite(LED,LOW); //Mandamos un BAJO al LED.
 }

}
```

Circuito:



Explicación:

Podemos ver en nuestro circuito que hemos conectado un led al pin 2 y un pulsador al pin 3, en la programación definimos un alias para cada uno respectivamente usando **#define**, además que al led lo ponemos como salida usando **pinMode()** , al pulsador como entrada ya que leeremos lo que mande, este pulsador lo hemos conectado en modo de **pull-down**, esto nos quiere decir que siempre enviará un estado lógico de **BAJO** estando en reposo y cuando pulsemos enviará un **ALTO**, este valor se almacenará en la variable **Push_lee**, pero para leer la señal que el pulsador manda usamos la función **digitalRead()**, ese valor se almacena en la variable anteriormente mencionada, todo esto dentro de loop ya que es nuestro programa de ejecución, después con el uso de una condicional **if** tendremos la lógica **"Si presionamos el pulsador este enviará un valor lógico ALTO o 1, entonces prende el led si no se pulsa, entonces apaga el led"**, por eso el if si tenemos un valor **ALTO** o **1** en nuestra variable **Push_lee** y hacemos la comparación, esto ya se ha explicado en el primer capítulo, entonces entra dentro del if y envía un alto al **LED** y este prenderá, si no entra en el **else** ya que no se tiene un **uno** de lectura sino un **cero**, entonces envía un bajo al **LED** y este permanece apagado únicamente si no se pulsa el pulsador. Y es así como leemos valores digitales.

En la siguiente práctica veremos cómo usar estas lecturas con un poco de lógica de programación para poder hacer que un led permanezca prendido, cabe mencionar que no es necesario que sea un pulsador el que mande la señal como lo veremos más adelante, si no puede ser cualquier sensor que solo envíe dos estados, **ALTO** o **BAJO**.

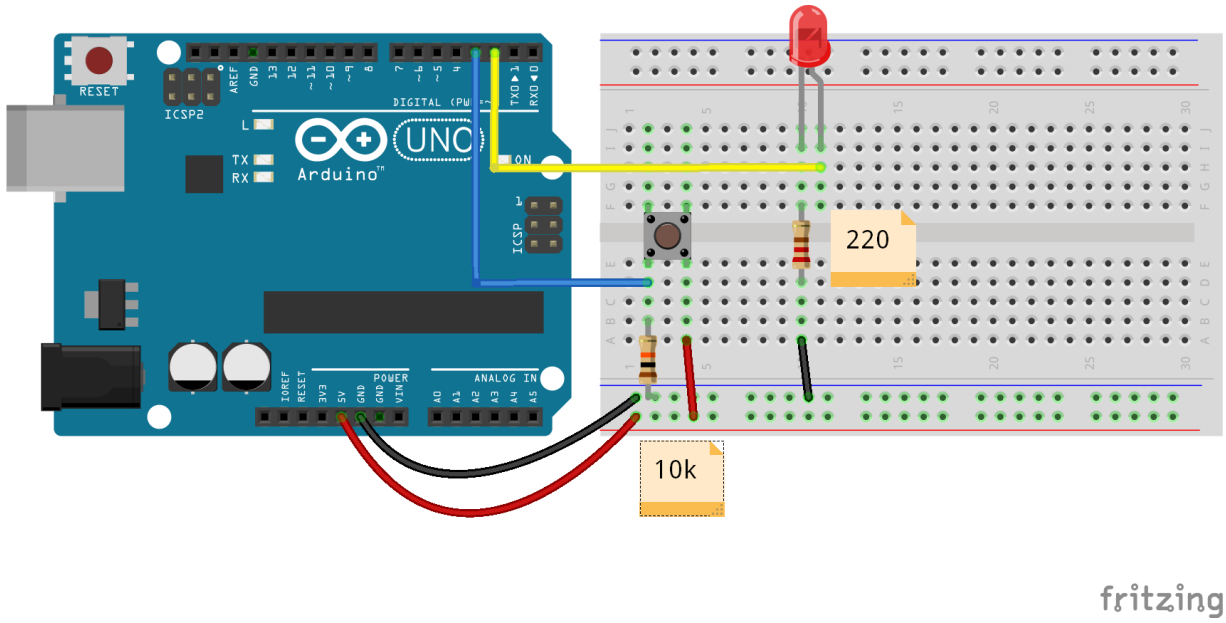
4.1.2 PULSADOR COMO INTERRUPTOR

Como se mencionó anteriormente se hará una práctica, la cual consta de que al momento de presionar un pulsador este prenderá un led hasta que se vuelva a presionar, convirtiendo este pulsador a un interruptor con programación, esto no solo nos servirá con pulsadores sino con muchos otros componentes, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
1 Pulsador de 4 pines o 2 pines
1 Resistencia de 10k
Protoboard
Jumpers

```
#define LED 2
#define push 3

int push_lee = 0;
int estadoLed = 0;
int estadoAnterior = 0;

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(push, INPUT);
}

void loop() {
  push_lee = digitalRead(push);
  if ((push_lee == 1) && (estadoAnterior == 0)) {
    estadoLed = 1 - estadoLed;
    delay(10);
  }
  estadoAnterior = push_lee;

  if (estadoLed == 1) {
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }
}
```

Circuito:



Explicación:

Como pueden ver el código que hemos escrito ya no tiene comentarios y esto se eliminó porque ya hemos avanzado y aprendido cómo funcionan las cosas básicas, así como definir y darle alias a los pines y ponerlos como entrada o salida, lo cual hacemos con el pin 3 y pin 2 respectivamente.

Creamos tres variables del tipo int, una para almacenar la lectura del pulsador que es **push_lee**, **estadoLed** para almacenar el estado que entrará en el if para que prenda o apague el led, incluso cualquier otro actuador que veremos más adelante.

La variable **estadoAnterior** almacenará el estado anterior del pulsador, estas tres variables se igualan a cero para tener un valor por defecto, ya en el **setup()** se configuran los pines como entrada y salida respectivamente, ahora viene lo más interesante que es nuestro programa de ejecución que sirve para usar un pulsador como un switch.

Primero almacenamos en **push_lee** la lectura que se le hace al pulsador, ahora escribiremos un if en el cual tendremos dos condiciones unidas por un **AND** o sea un **&&**, la primera **condición** nos dice que al estado de **push_lee** debe ser **uno** y se hace la comparación además de que este estado debe ser **verdadero** y esto se cumple cuando presionamos el pulsador, la segunda **condición** dice que el estado anterior debe ser **cero** y esto es verdad ya que **al principio** nosotros le dimos un valor de **cero**, después tiene una operación muy interesante la cual es **estadoLed = 1 – estadoLed**, esta se explicará más adelante.

Después dentro del **if** viene un **delay()**, es un pequeño tiempo llamado **antirrebote** y es el tiempo en que tardamos presionando el pulsador, al terminar este if se le asigna a la variable **estadoAnterior** el valor de **push_lee**.

Ahora bien, explicaremos la operación **estadoLed = 1 – estadoLed**, como el pulsador están en modo **pull-downn** al presionar nosotros el pulsador el valor de la variable **push_lee** cambia a **uno**, después entra al **if** donde las **dos condiciones** se evalúan y se encuentran verdaderas ya que se cumple la primera al presionar el pulsador y la segunda se cumple pues le dimos el valor de cero, por consiguiente entra al if donde tenemos la operación **estadoLed = 1 – estadoLed**.

Como la variable **estadoLed** tiene valor **cero** porque así lo hemos puesto al principio entonces tenemos que **estadoLed = 1 – 0**, eso nos da el valor de **uno** en **estadoLed**, y ahora en la variable **estadoAnterior** fuera del **if** tiene un valor **cero** ya que el valor de **uno** lo toma **únicamente** cuando presionamos el pulsador y cuando se suelta toma el valor de **cero** nuevamente, después de esto entra el **segundo if**, como el **estadoLed** vale **uno** y no se ha cambiado este valor permanece en ese **if** y **prende el led** hasta que se vuelva a presiona el pulsador cambiará de estado porque ahora se tendrá: **estadoLed = 1 – 1**, que es **cero** y entra en el **else** ya que no se cumple la primera condición y se apaga el led, y así es como se hace para que un pulsador funcione como switch, como ya se mencionó esto no sirve solo para pulsadores, sino para muchas más aplicaciones que se verán más adelante.

Muy bien, ahora ya saben controlar mejor las señales de entrada con programación.

4.1.3 OPTOINTERRUPTOR

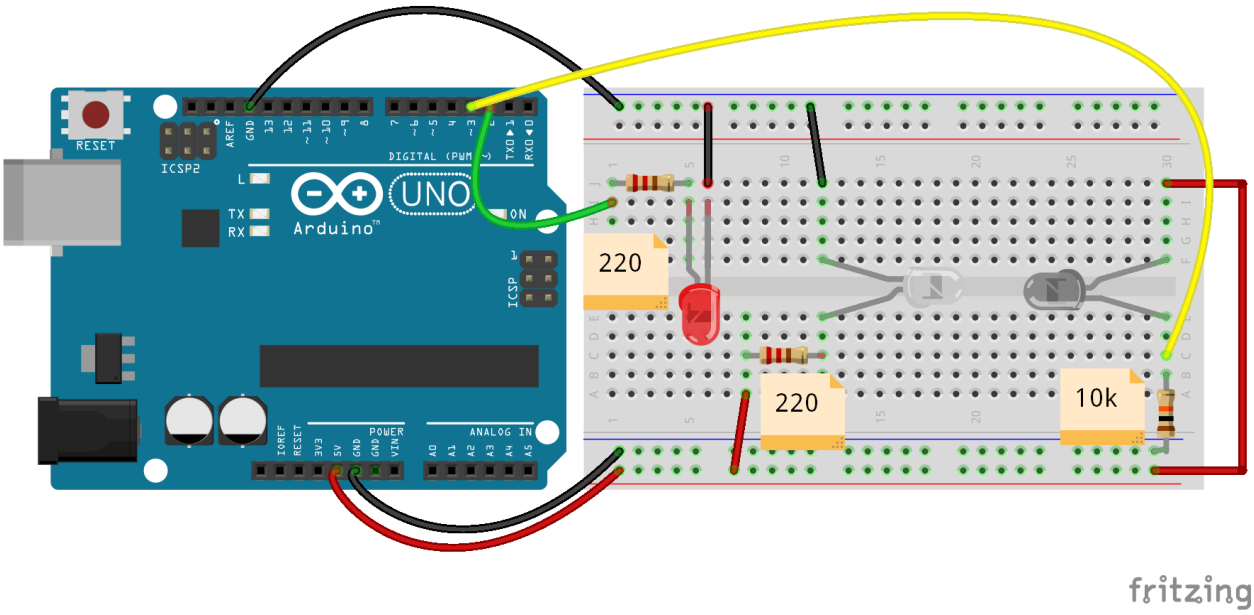
Ya vimos como leer sensores digitales, ahora haremos una práctica bastante simple que será la base para hacer muchos proyectos, el cual es saber manejar y leer los datos que nos manda un optointerruptor, estos se usan en las cajas del supermercado en la banda trasportadora que al bloquear la señal se detiene pero si es una señal limpia avanza la banda o también para tener un conteo se personas u objetos, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
2 Resistencia de 220Ω
1 Diodo emisor de 5mm IR
1 Resistencia de 10k
1 Fototransistor
Protoboard
Jumpers

```
#define LED 2
#define Opto 3
int Opto_lee;

void setup() {
  pinMode(LED, OUTPUT);
```

```
  pinMode(Opto, INPUT);
}

void loop() {
  Opto_lee = digitalRead(Opto);

  if (Opto_lee == 1) {
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }

}
```

Circuito:



Explicación:

Muy bien, el funcionamiento es muy simple, definimos dos pines, el led y el optointerruptor al led lo ponemos como **OUTPUT** y para el optointerruptor lo ponemos como **INPUT**, también creamos una variable del tipo entero que almacenará el valor enviado por el fototransistor al **pin 3**, después en el loop hacemos nuestro programa de ejecución, usando **digitalRead** leemos el valor del **pin 3** y lo almacenamos en la variable **Opto_lee**, después con un if hacemos una comparación lógica, si el valor digital es 1 enciende el led usando **digitalWrite**, en caso de que sea un valor distinto a 1 permanecerá apagado.

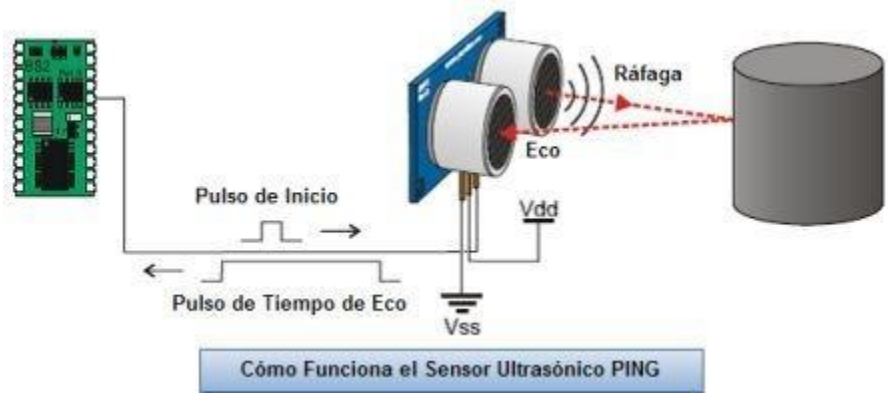
Como vimos este es el funcionamiento de un optointerruptor que nos servirá para hacer muchos proyectos, desde una banda trasportadora hasta un seguidor de líneas.

4.1.4 OTROS SENSORES DIGITALES

Ya vimos los sensores más simples que solo envían una señal por acción ya sea un 1 o un 0 digital, ahora veremos otros sensores digitales que funcionan similarmente. Este ejemplo se hará con el sensor ultrasónico HC-SR04, aquí se verán una cuantas funciones nuevas que se explicarán más adelante cuando se explique el funcionamiento del código, abajo se muestra el código, material y explicación de la práctica.

Explicación del ultrasónico:

El funcionamiento interno del sensor ultrasónico es bastante complejo, pero explicaremos lo más simple. El sensor envía un pulso durante un tiempo (especificado en la hoja de datos), ese pulso sale por medio de un piezo que genera una onda ultrasónica, ésta choca con algún objeto y rebota, esta misma onda viaja a través del aire y es recibida por el otro "piezo" del sensor llamado eco y usando formulas bastantes simples con algún microcontrolador, en esta caso Arduino, podremos saber la distancia a la cual se encuentra el objeto en que la onda rebotó, se puede apreciar un ejemplo gráfico en la imagen de abajo.



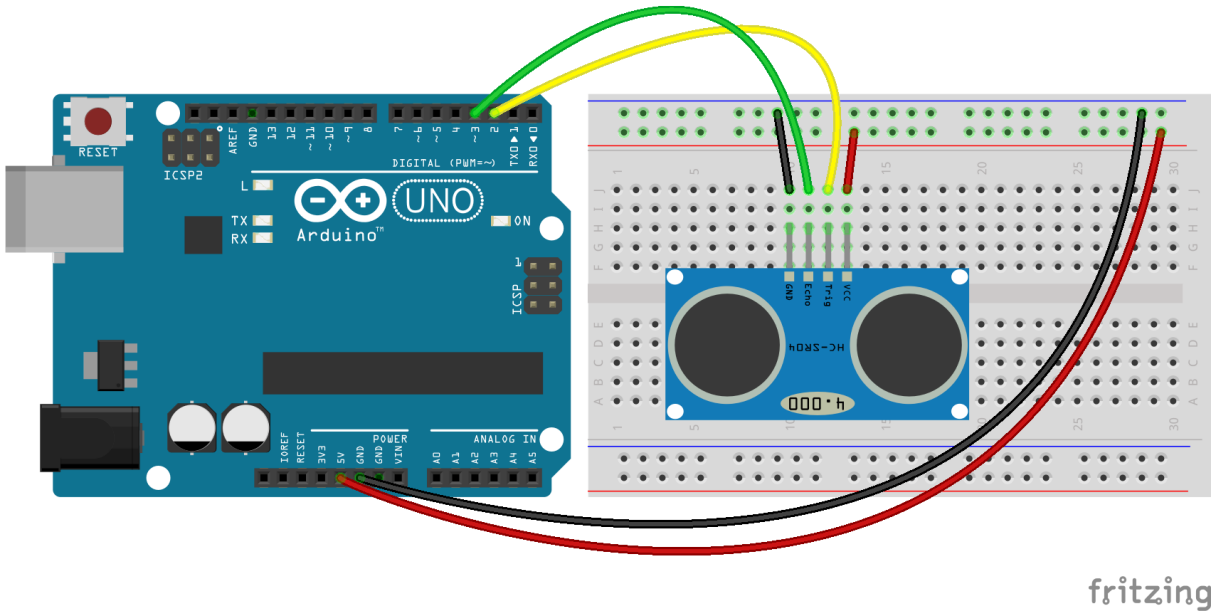
Todos estos datos se encuentran en la hoja de datos del sensor, se recomienda buscarla y leerla detenidamente.

MATERIALES
1 ARDUINO
1 HC-SR04
Protoboard
Jumpers

```
#define TRIG 2
#define ECHO 3

long Tiempo;
float Distancia;

void setup() {
```

```
  Serial.begin(9600);
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
}

void loop() {
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  Tiempo = (pulseIn(ECHO, HIGH)/2);
  Distancia = float(Tiempo * 0.0343);
  Serial.println(Distancia);
  delay(1000);
}
```

Circuito:



Explicación:

Ahora explicaremos cómo funciona el código junto al sensor para poder medir la distancia.

Primero definimos los dos pines del sensor **TRIG** y **ECHO**, después definimos dos variables, en una se almacenará el tiempo de regreso de la onda ultrasónica que viaja por el aire y en la otra almacenará la distancia. En el setup ponemos los pines tanto de entrada y salida respectivamente, además escribimos **Serial.begin(9600)**, esto nos servirá para iniciar la comunicación serial para poder ver la distancia, más adelante se explicará la comunicación Serial con profundidad.

Como dice la hoja de datos del sensor, necesitamos enviar un pulso por 10 microsegundos por el **TRIG**, ahora lo que hacemos es usar **digitalWrite** y enviar un pulso alto por el pin del **TRIG** por 10us, esto se hace mediante el **delayMicroseconds**, después apagamos ese pulso con un **LOW**.

Después, como se dijo en la explicación del ultrasónico esa onda regresará y será leída por el ECHO, para esto usaremos la función **pulseIn**, esta función lo que hace es contar el tiempo en que hay un pulso, ya sea alto o bajo, este tiempo se divide entre 2 ya que lo dice la hoja de datos de nuestro sensor, esto es el tiempo que tarda el sonido en ir y volver.

Ahora calcularemos la distancia, sabiendo que el espacio es igual a la velocidad por el tiempo y que la velocidad del sonido es de 343m/s donde el tiempo lo tenemos en millonésimas de segundo, hacemos la operación matemática de multiplicar el tiempo por la velocidad del sonido y esto nuevamente viene en la hoja de datos del sensor, después simplemente ese valor se almacena en la variable Distancia.

Después con la función **Serial.println**, imprimimos el valor de la distancia, ahora solo queda abrir el monitor serial en **Herramientas > Monitor Serie**.

Bien, con esto terminamos el tema de sensores digitales, ya sabes leer señales digitales se 0 o 5v, a continuación veremos el uso de sensores analógicos.

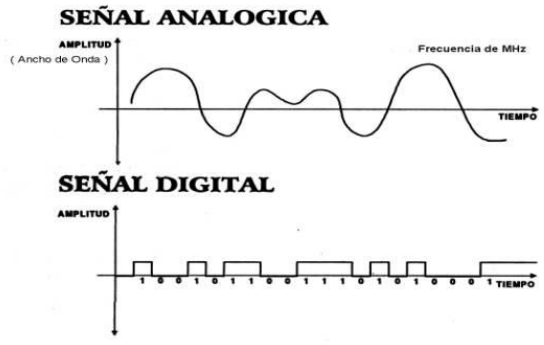
Video **[tutorial]:** *[Curso Arduino Básico - #4] Sensores Digitales*

4.2 SENSORES ANALÓGICOS

Ya vimos el uso de los sensores digitales y aprendimos como recibir señales digitales de 0 o 5v, ahora veamos cómo leer señales analógicas con nuestro Arduino.

- Digital:
  - Solo tiene dos valores 0 o 5 volts.
  - Ejemplo: 0 - 1, HIGH – LOW.
- Analógico:
  - Valores continuos en un rango.
  - Ejemplo: Todos los valores entre 0 y 5 volts.

En la siguiente imagen tenemos una representación gráfica de valores digitales contra valores analógicos.



Como bien ya explicamos Arduino es un dispositivo digital, pero ahora leeremos valores analógico que se encuentran en rangos continuos y es necesario convertir el valor analógico a digital, pero esto Arduino lo hará por nosotros.

- Arduino UNO tiene 6 puertos analógicos:
    - **A0** al **A5**.
    - Son puertos de entrada analógica y/o entrada y salida digital.
    - El microcontrolador tiene un transductor que convierte la señal analógica a digital.

La entrada del valor analógico tiene que estar entre 0 y 5 volts, gracias al transductor los valores continuos entre 0 y 5 voltios son convertidos a un número entero, gracias al bootloader Arduino está pre configurado a un resolución de **10 bits**, eso quiere decir que, si tienes 0 volts tendremos un valor de 0, y si tenemos **5 volts** tendremos un valor de **1023**, esto nos quiere decir que el valor de 2.5 volts es 512.

Podemos cambiar el valor del voltaje de referencia por un valor menor a 5V conectando ese valor al pin **AREF** y con la función **analogReference()**, y esto nos sirve por ejemplo si tienen un sensor analógico que funciona a **3.3 volts**.

Para poder sensar los valores analógicos ya comentamos debemos de conectar el sensor a cualquier pin desde A0 al A5 y ya en el código para leer esos valores tendremos que usar lo siguiente:

- Para leer un pin analógico usamos:
    - analogRead([numpin/alias]);
    - Nos regresa un valor entre 0 y 1023.
    - Usamos la comunicación serial para ver su valor.

Como se mencionó nos regresará un valor entre 0 y 1023, esto gracias a la conversión de 10 bits. En el tema anterior vimos el uso del Serial para ver los datos y lo volveremos a hacer con los sensores analógicos.

A continuación se verán algunas prácticas que como ya se mencionó serán la base para realizar muchos proyectos uniendo lo ya visto.

4.2.1 POTENCIÓMETRO

En esta primera práctica veremos cómo sensar un potenciómetro, esto nos ayudará más adelante a controlar el pwm y también a mover servomotores con un potenciómetro, así que esto será la base para los sensores analógicos y su uso, abajo se muestra el código, material y explicación de la práctica.

Primero explicaremos un poco cómo funciona la conversión analógico digital.

Como bien se comentó la resolución de Arduino es de 10 bits eso quiere decir que tenemos un valor máximo de 1024, esta resolución es 2^n con $n = 10$, entre una mayor resolución más exacta será el resultado del sensor.

Por ejemplo, si tenemos un voltaje de 2.5 volts tendremos un valor en resolución de 10 bits de 512, se preguntarán como sabemos eso, bien tenemos la siguiente formula:

$$V = \frac{ValAnalog * Vref}{2^n - 1}$$

Si despejamos al formula tendremos:

$$ValAnalog = \frac{V * (2^n - 1)}{Vref}$$

Como se comentó el voltaje de referencia de Arduino por default es de 5 volts, la resolución es de 10 bits o sea $2^{10} = 1024$, y queremos saber el valor para 2.5 volts, entonces tendremos:

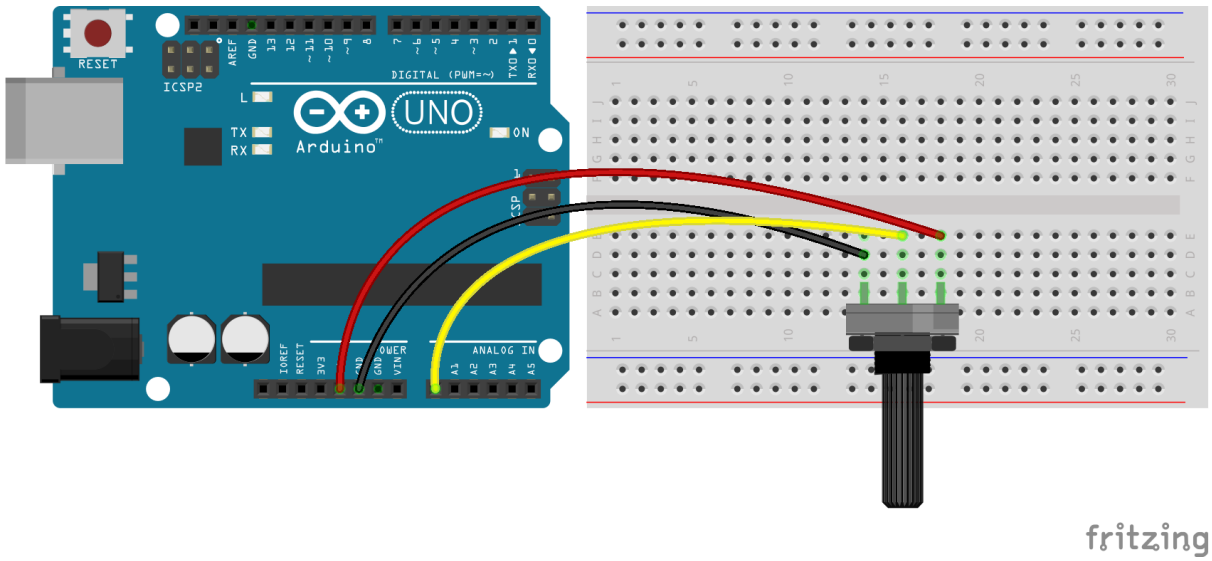
$$ValAnalog = \frac{2.5 * (1024 - 1)}{5.0} = 511.5 \approx 512$$

Esto es lo que Arduino hace internamente para realizar la conversión analógico/digital, ahora bien para saber el voltaje aplicamos la primera fórmula, esto nos servirá para el uso de muchos sensores, claro aplicando lo que dice la hoja de datos del sensor, esto se verá más adelante con el LM35.

MATERIALES
1 ARDUINO
1 Potenciómetro de 10k
Protoboard
Jumpers

```
#define POT A0
int LeePot;


void setup() {

  Serial.begin(9600);
  pinMode(POT,INPUT);

}

void loop() {

  LeePot = analogRead(POT);
  Serial.println(LeePot);
  delay(50);

}
```

Circuito:



Explicación:

La explicación es bastante simple, primero definimos los pines que usaremos y las variables para almacenar los valores, esto ya se ha hecho muchas veces anteriormente así que no se enterará en detalles.

Después en el **setup** inicializamos el pin como entrada, muchos no hacen eso ya que por default los pines analógicos ya están configurados como entrada, pero recordemos que tenemos que aprender a programar limpia y correctamente, también cargamos el **Serial** para la comunicación y poder ver el valor del potenciómetro.

Dentro del **loop** usamos la función **analogRead** para poder leer los valores analógicos del pin donde hemos conectado la salida del potenciómetro, ese valor se almacena en la variable **LeePot**, una vez hecho eso imprimimos el valor usando **Serial.println** pasándole la variable **LeePot** como parámetro, solo queda por abrir el monitor serial para visualizar los valores que aparecen al mover el potenciómetro.

Estos valores están en la resolución de 10 bits, para ver el voltaje que nos da simplemente aplicamos la primera fórmula que vimos e imprimimos los valores.

Con esto queda por visto como leer valores analógicos que van de 0 a 5 volts, en la siguiente práctica se muestra cómo aplicar lógica de programación además de matemáticas para poder obtener los datos deseados.

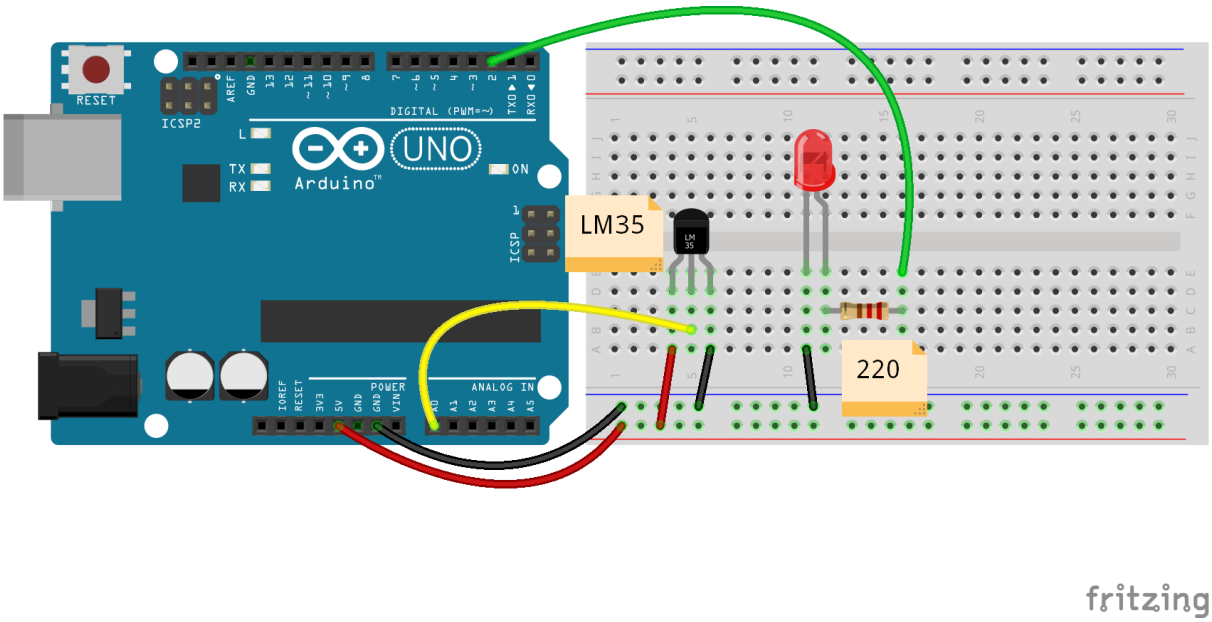
4.2.2 SENSOR DE TEMPERATURA LM35

Muy bien, ahora veremos cómo sensar un LM35 y cuando llegue a una temperatura hacer que prenda un led, esto servirá para controlar la temperatura de algún objeto y cuando llegue a un temperatura alta accione un ventilador o algún otro actuador, es muy importante que lean la hoja de datos del LM35 ya que allí viene descrito su funcionamiento, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Potenciómetro de 10k
Protoboard
Jumpers

```
#define LM35 A0
#define LED 2
float Temperatura;
float ValSen;

void setup() {

  Serial.begin(9600);
  pinMode(LM35,INPUT);
  pinMode(LED,OUTPUT);

}

void loop() {

  ValSen = analogRead(LM35);
  Temperatura = (ValSen * 5.0 * 100.0) / 1023.0;

  if(Temperatura > 35){
    digitalWrite(LED,HIGH);
  }else{
    digitalWrite(LED,LOW);
  }

  Serial.println(Temperatura);
  delay(50);

}
```

Circuito:



Explicación:

Primero definimos los pines que usaremos y las variables para almacenar los valores, esto ya se ha hecho muchas veces anteriormente, se definen dos variables, una donde almacenaremos los valores del ADC y otra para almacenar la temperatura.

Como bien se mencionó anteriormente, se recomienda leer la hoja de datos del sensor, ya que aquí viene una información muy importante la cual nos dice que nos dará 10mV por cada grado centígrado, esto es fundamental para el funcionamiento, veamos un ejemplo.

Pensemos que tenemos la temperatura de 20.5 grados centígrados, entonces esto nos dice que hay 205mV como resultado de $10mV * 20.5 = 205mV$, ahora hacemos la conversión a voltaje que son 0.205 voltios y ahora aplicaremos nuestras fórmulas anteriormente vistas:

$$ValAnalog = \frac{0.205 * (1024 - 1)}{5.0} = 41.943$$

Así obtenemos el valor analógico que es la conversión de 10 bits equivalente a 41.943, ahora aplicando la primera fórmula pero multiplicando por 100 para convertir el resultado a grados, entonces nos queda:

$$T = \frac{41.943 * 5.0 * 100.0}{1024 - 1} = 20.5°$$

Recordemos que 5.0 es el valor del voltaje de referencia o sea 5 voltios, 41.943 el valor analógico que entregará el ADC y bueno lo demás ya se ha explicado con anterioridad.

Así es como obtenemos la temperatura usando el sensor LM35, solo aplicamos estos cálculos en la programación, usamos **analogRead** para leer el valor que envía el sensor al Arduino y lo almacenamos en la variable **ValSen**, después aplicando la **fórmula anterior** obtendremos la temperatura y esta la almacenamos en la variable **Temperatura**, una vez hecho eso creamos una condicional que si la temperatura llega a ser más de 35 grados se prenderá el led o lo que ustedes quieran accionar, en caso contrario el led permanece apagado, después imprimimos la temperatura por medio del **Serial** y la visualizamos en nuestro **monitor serial**.

4.2.3 SENSOR IR TCRT5000

Ya vimos como leer sensores analógicos e incluso aplicar los datos de sus hojas de datos para poder hacerlos funcionar correctamente, ahora la última práctica de sensores analógicos será censar con el sensor TCRT5000 el cual es un sensor infrarrojo, estos sensores son usados en los seguidores de líneas para distinguir del color blanco y negro.

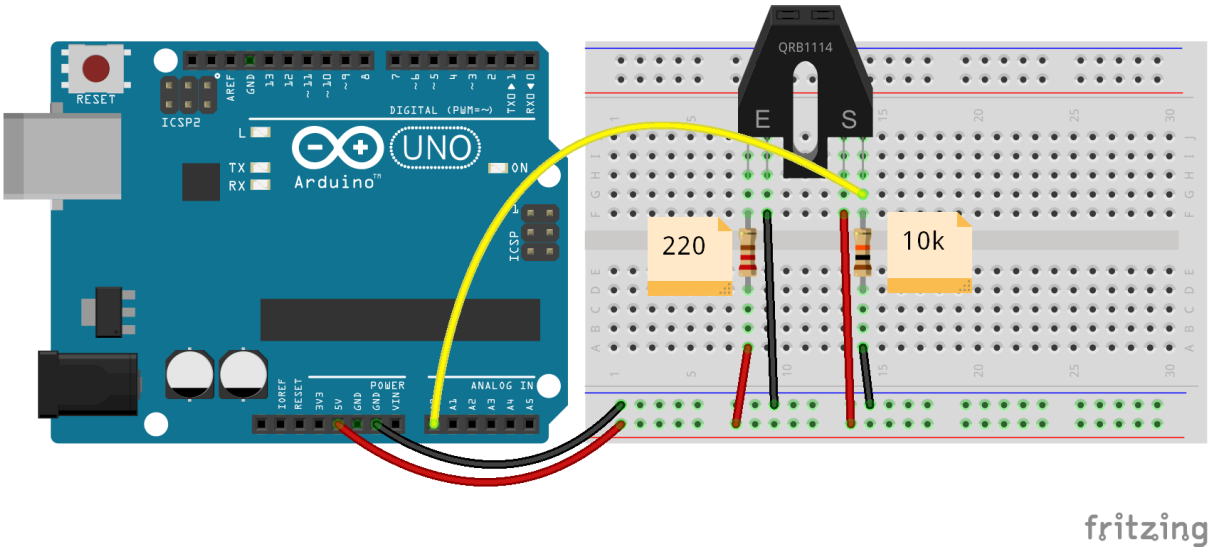
Los valores de estos sensores nunca serán iguales aunque sean hechos por la misma empresa, un sensor puede dar el valor de 800 y otro de 900 cuando detecten el color blanco, aclaro que esto es un ejemplo, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Resistencia de 10k
1 Resistencia de 220 Ohms
1 TCRT5000
Protoboard

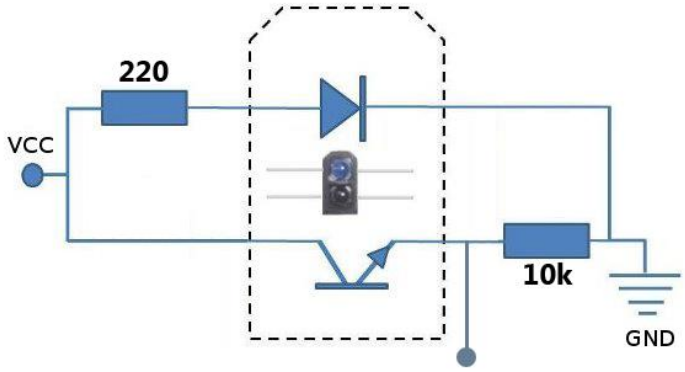
```
#define TCRT A0

int valor;

void setup() {
  Serial.begin(9600);
  pinMode(TCRT, INPUT);

}

void loop() {
  valor = analogRead(TCRT);
  Serial.print("Valor: ");
  Serial.print(valor);
  Serial.print(" | Color: ");
  if (valor < 800) {
    Serial.println("Blanco");
```

```
  } else {
    Serial.println("Negro");
  }
  delay(500);
}
```

Circuito:



Referencia del sensor:



Explicación:

El sensor se conecta en modo pull-down, también se puede usar como digital pero esta vez lo empleamos como analógico, al hacer la conexión como pull-down creamos un divisor resistivo y obtenemos un valor dependiendo de cuanta luz recibe del emisor, y todo esto se envía al pin análogo A0 de nuestro Arduino.

El código es bastante simple, definimos el pin A0 como entrada, creamos una variable del tipo entero para almacenar los datos del sensor. En el setup inicializamos el pin como entrada y cargamos el serial para visualizar los datos.

En el **loop** usamos **analogRead** y ese valor lo almacenamos en la variable **valor**, usando el **Serial.println** imprimimos el valor que nos arroja el sensor además el "color" que ve, lo cierto es que no ve ningún color, sino que el blanco o negro refleja más o menos luz IR del emisor que llega al fototransistor.

Después, por medio de una condicional if y un valor de comparación random en este caso 800, se identificará de qué color se trata, si es menor de 800 imprimirá que es blanco, en otro caso será negro.

Con esta última práctica queda finalizado el tema de sensores analógicos, como se vio, solo es aplicar fórmulas además de seguir las indicaciones de las hojas de datos de nuestros sensores analógicos a usar, en el siguiente tema veremos actuadores analógicos que serán pwm, que junto con manejo de cargas más adelante podrán controlar muchísimas cosas.

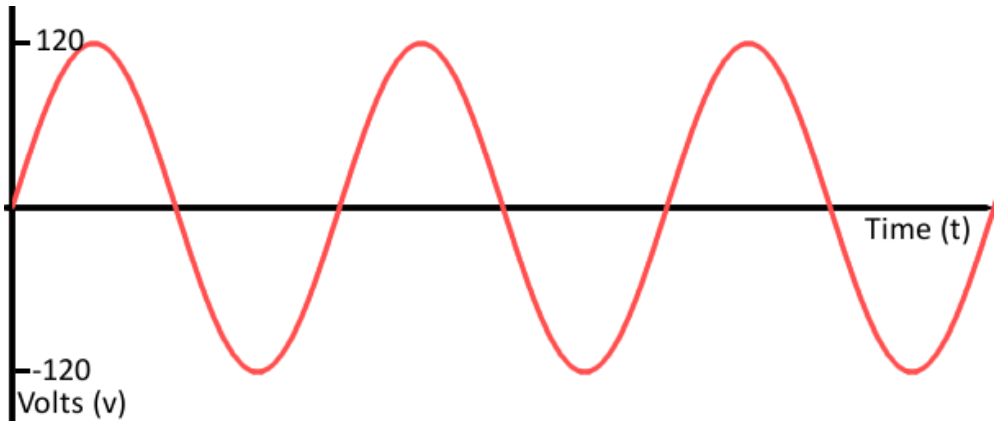
Video [tutorial]: *[Curso Arduino Básico - #5] Sensores Analogicos*

4.3 ACTUADORES ANALÓGICOS

Ya vimos como censar valores analógicos, pero ahora veremos como "escribir" valores analógicos, o sea enviar valores analógicos.

¿Qué es algo analógico?

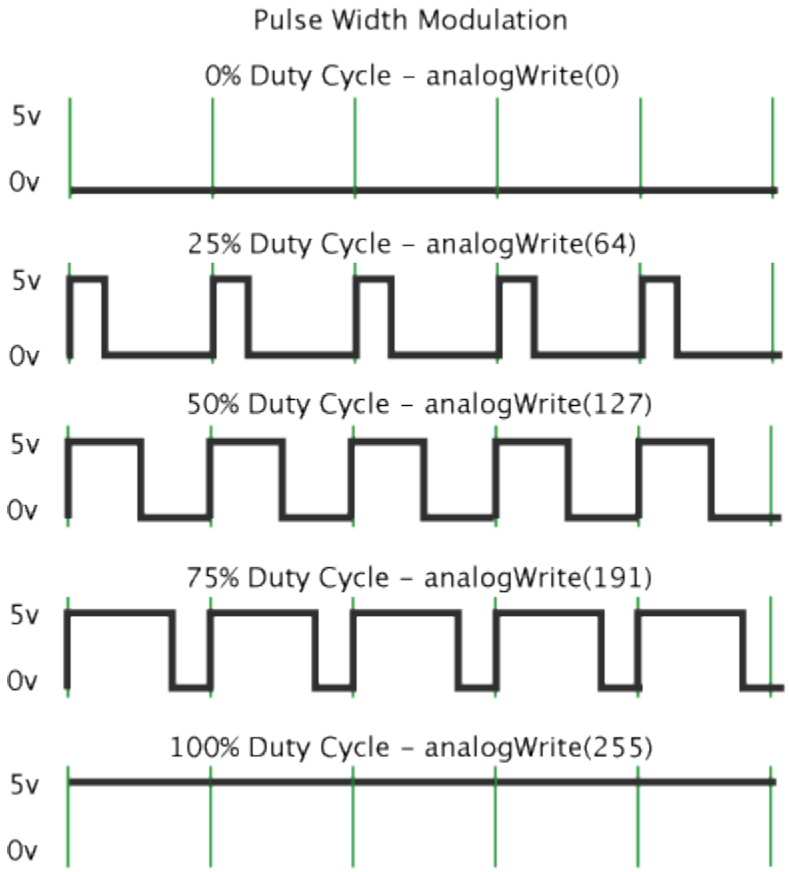
- Analógico:
  - Valores continuos en un rango.
  - Ejemplo: Todos los valores entre 0 y 5 volts, esto en resolución de 10 bits serán valores entre 0 a 1024.



Arduino no tiene salidas propiamente analógicas ya que los microcontroladores son dispositivos digitales.

- Se utiliza PWM, Modulación por Ancho de Pulso que es:
    - Una señal cuadrada entre 0 y 5 volts.
    - Esta señal cambia rápidamente entre esos dos valores.
    - Simula voltajes analógicos entre 0 y 5 volts.
    - Tiene una resolución de 8 bits.

En la siguiente imagen se muestra una gráfica de los ciclos de PWM:



El Arduino UNO tiene 6 salidas PWM, las cuales son los pines 3, 5, 6, 9, 10, 11 y están marcados con PWM o con ~.

Para escribir con estos pines usamos la función analogWrite([pin/alias],[valor/variable]);, en valor o variable como bien comentamos solo recibe valores que estén entre 0 y 255.

A continuación haremos unas prácticas que serán como siempre se ha mencionado, la base para poder hacer proyectos más avanzados como por ejemplo control de luces RGB o intensidad de luminiscencia.

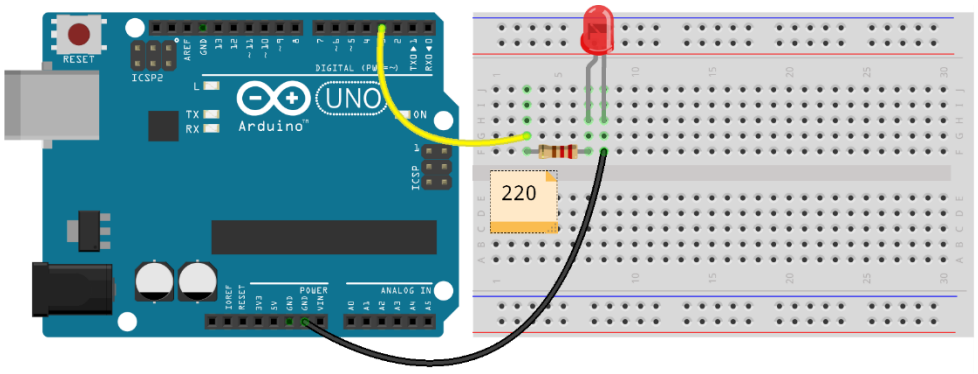
4.3.1 FADE DE UN LED

Ya hemos visto sensores analógicos, ahora en esta práctica enviaremos valores analógicos para poder controlar la luminiscencia de un led, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
Protoboard
Jumpers

```
#define LED 3

void setup() {
 pinMode(LED,OUTPUT);

}

void loop() {
  for(int i = 1; i <= 255; i++){
    analogWrite(LED,i);
    delay(10);
    }
    delay(1000);
  for(int i = 255; i >= 1; i--){
    analogWrite(LED,i);
    delay(10);
    }
  delay(1000);
}
```

Circuito:



Explicación:

El funcionamiento es muy simple primero se define el pin para el LED, después en el setup se pone el LED como salida, en el **loop** se crea un **for** que ira desde 0 a 255, recordemos que la resolución del PWM es de 8 bits.

Dentro del for usamos **analogWrite**, donde recibe como parámetro el **LED** y los valores del **for** que ira incrementando de uno en uno y esto se almacena en el la variable **i**, se le da un delay de 10, ósea que cada 10 milisegundo se ira incrementando la variable **i**.

Se da un segundo delay de 1000 que será el tiempo en que se le da una pausa al LED, para que cambie el asegundo **for**, que hará casi lo del primer **for**, pero esta vez decrementa ósea, que empieza desde el 255 y termina en 0, recordamos que la sentencia **for** se vio en capítulos anteriores.

Gracias a los dos **for** vemos como incrementa y decrementa el brillo del led, si usamos un multímetro, veremos cómo aumenta y disminuye el voltaje entre los rangos de 0 a 5 volts.

Y es así como manejamos el PWM, lo que hace es que envía pulsos muy rápidos y eso hace que se emule un valor analógico y eso se hace como ya vimos con **analogWrite**.

En la siguiente práctica se unirá lo que se vio en sensores analógicos.

4.3.2 FADE DE UN LED CON POTENCIOMETRO

Ya se vio como aumentar o disminuir el brillo de un led con PWM y con el uso de una sentencia for, ahora lo haremos usando un potenciómetro, la explicación del circuito será muy simple y vaga ya que la mayoría de lo que se usa se vio en temas anteriores.

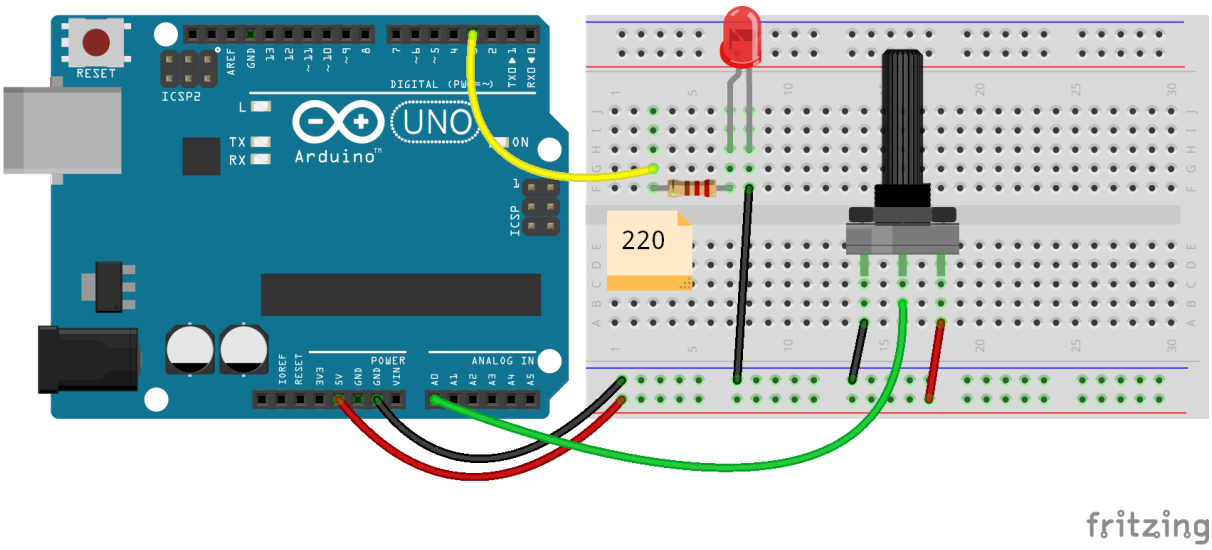
Como en sensores analógicos ya se vio como leer los valores de un potenciómetro y almacenarlos en una variable para su uso posterior y en la práctica anterior ya se vio el uso de PWM con **analogWrite**, recuerde que para usar el PWM solo se deben de conectar los actuadores en los pines mencionados al principio de este tema, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
1 Potenciometro de 10k
Protoboard
Jumpers

```
#define LED 3
#define POT A0
```

```
void setup() {
  Serial.begin(9600);
  pinMode(POT,INPUT);
  pinMode(LED,OUTPUT);
}

void loop() {
  int valor = analogRead(POT);

  Serial.print("Valor POT: ");
  Serial.print(valor);
  Serial.print(" | Valor PWM: ");
  Serial.println(valor/4);

  analogWrite(LED,valor/4);
  delay(100);
}
```

Circuito:



Explicación:

La explicación es bastante simple, se definieron los pines a usar, en este caso el **3** y **A0,** se creó una variable para almacenar los datos del potenciómetro, en el **setup** inicializamos los pines y cargamos el **Serial** para visualizar los datos.

En el **loop** leemos los datos del potenciómetro y se almacenan en la varíale **valor**, después con el uso del Serial.println imprimimos esos valores. Y se preguntarán, ¿por qué se divide entre 4?, es muy simple, recuerden que la resolución del Arduino para el ADC es de 10 bits así que nos entregará un máximo de 1024 - 1, pero la resolución del PWM es de 8 bits y acepta un valor máximo de 255, así que $\frac{1023}{4} = 255$,

por eso se divide entre 4 para obtener el valor máximo de la resolución del PWM. Esto también se hace en la función **analogWrite**, pues es ahí donde enviaremos esos valores al **LED** para que aumente y disminuya su brillo cada 100 milisegundos por medio del potenciómetro, esto con ayuda del delay.

Ahora pueden mover el potenciómetro y podrán apreciar cómo cambia el brillo del led, además de visualizar los datos en el monitor serial.

Con esto vemos cómo podemos manipular el PWM con sensores, en este caso un potenciómetro y como siempre esto es la base para proyectos más avanzados.

4.3.3 FADE DE UN LED CON POTENCIÓMETRO Y LA FUNCIÓN MAP

Ya vimos cómo usar sensores analógicos y manipular actuadores analógicos por medio del PWM con la función analogWrite y un potenciómetro, pero claro esto lo hicimos de la manera más simple, ahora lo haremos con la función **map**, esta función es muy importante y nos ayudará mucho en los proyectos que queramos desarrollar, aquí una pequeña explicación de la función **map**.

En palabras simples, la función **map** hace una conversión de valores entre un rango dado por una variable a valores de otro rango, por ejemplo:

```
y = map(x, 0, 1023, 0, 255);
```

En este ejemplo se está empleando la función **map**, en la cual como **primer parámetro** recibe la variable de los datos a convertir, después recibe el primer valor **mínimo** que se tendrá en la variable **x**, a continuación recibe el valor **máximo** que se tendrá en la variable **x**, como **cuarto parámetro** recibe el valor a convertir del valor **mínimo**, en este caso queremos que cero permanezca como cero y como **quinto parámetro** recibe el valor a convertir en valor **máximo** de la variable **x**, en este caso queremos que el **1023** sea **255**, esos valores se almacenarán en la variable **y**.

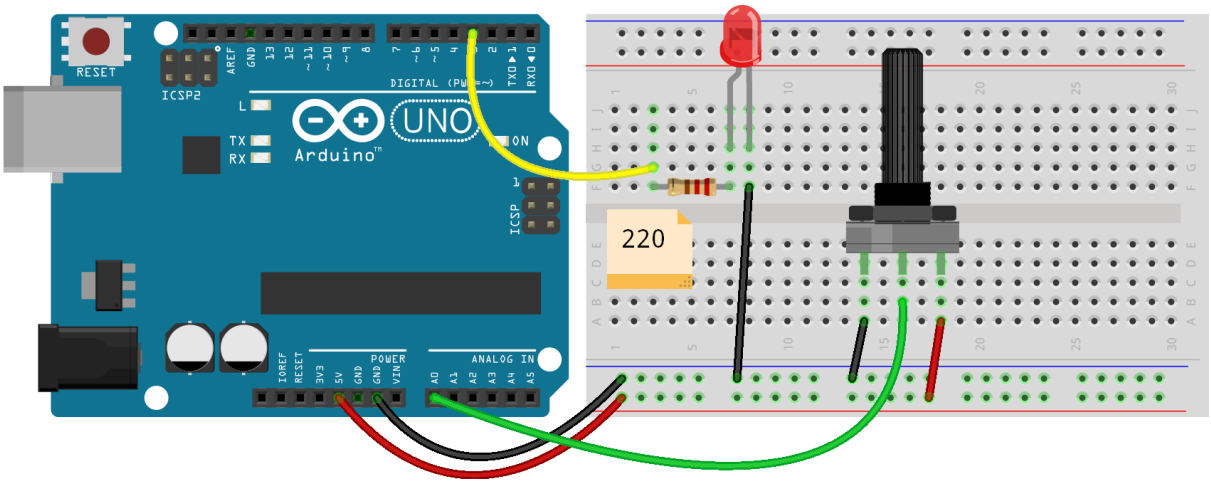
Se pueden convertir a cualquier tipo de valor en cualquier rango entero tanto positivo como negativo, más adelante se verán más ejemplos de esta función. Abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LED de 5mm
1 Resistencia de 220Ω
1 Potenciometro de 10k
Protoboard
Jumpers

```
#define LED 3
#define POT A0
```

```
void setup() {
  Serial.begin(9600);
  pinMode(POT,INPUT);
  pinMode(LED,OUTPUT);
}

void loop() {
  int valor = analogRead(POT);

  int mapeo = map(valor,0,1023,0,255);

  Serial.print("Valor POT: ");
  Serial.print(valor);
  Serial.print(" | Valor PWM: ");
  Serial.println(mapeo);

  analogWrite(LED,mapeo);
  delay(100);
}
```

Circuito:



Explicación:

El código es similar al anterior y la explicación también, lo único que se agrega en lugar de variables globales se usan **variables internas** del tipo entero en el **loop**, la primera almacena los datos del potenciómetro y la segunda almacenará los datos de la conversión de la función **map**, como se explicó anteriormente, la función **map** recibe la variable de la cual tomará los datos a convertir así como el

máximo y mínimo de estos valores dados por la variable valor y después los convertirá a valores queremos, en este caso de 0 a 255.

Después simplemente se imprimen los valores y a la función **analogWrite** también le pasamos estos valores almacenados en la variable **mapeo**.

En la siguiente práctica se ve cómo usar el PWM para manipular los valores de un led RGB que será la base para poder controlar tiras de led, esto claro uniéndolo con el tema de control de cargas que se verá más adelante.

4.3.4 CONTROL DE LED RGB

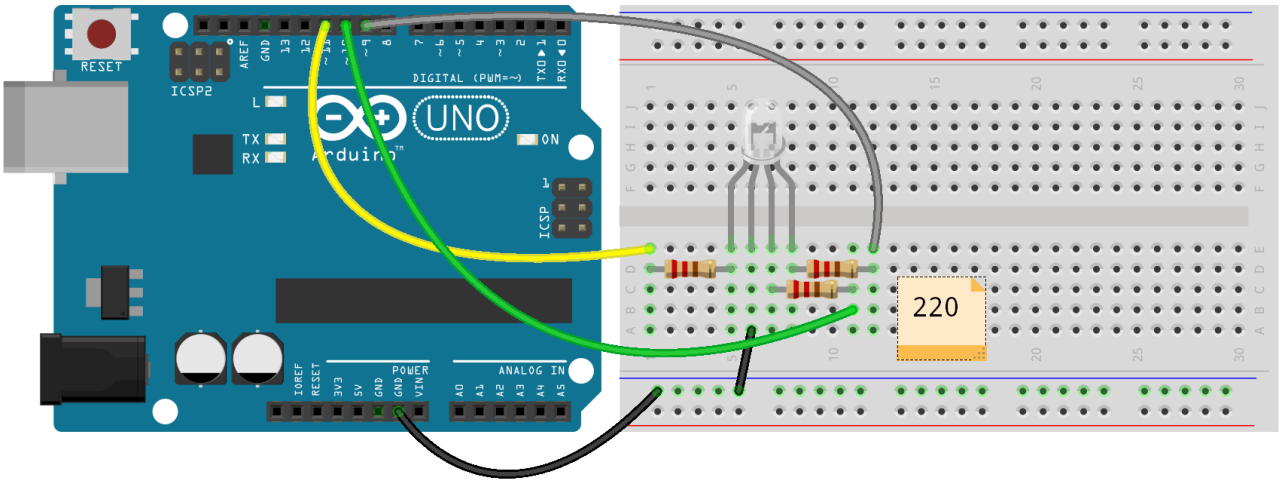
En esta práctica se implementa lo visto anteriormente para controlar un led RGB, esto puede ser usado con tiras de leds o leds RGB de alta potencia, claro queda uniendo este tema con control de cargas, este es un ejemplo para la base de lo ya comentado. Abajo se muestra el código, material y explicación de la de la práctica.

MATERIALES
1 ARDUINO
1 LED RGB de 5mm cátodo común
1 Resistencia de 220Ω
1 Potenciómetro de 10k
Protoboard
Jumpers

```
int pinLedR = 11;  // pin Rojo del led RGB
int pinLedV = 10;  // pin Verde del led RGB
int pinLedA = 9;   // pin Azul del led RGB

int pausa = 1000;

void setup() {
  pinMode(pinLedR, OUTPUT);    // pone el pinLedR como output
  pinMode(pinLedV, OUTPUT);    // pone el pinLedV como output
  pinMode(pinLedA, OUTPUT);    // pone el pinLedA como output
}

void loop() {
  //  colores basicos:
  color(255, 0, 0);   // rojo
  delay(pausa);       // delay por pausa
  color(0, 255, 0);   // verde
  delay(pausa);       // delay por pausa
  color(0, 0, 255);   // azul
  delay(pausa);       // delay por pausa
```

```
  // colores mezclados:
  color(255, 255, 255); // blanco
  delay(pausa);         // delay por pausa
  color(255, 255, 0); // amarillo
  delay(pausa);         // delay por pausa
  color(255, 0, 255); // magenta
  delay(pausa);         // delay por pausa
  color(0, 255, 255); // cian
  delay(pausa);         // delay por pausa
  color(0, 0, 0);       // apagado
  delay(pausa);         // delay por pausa
}

// funcion para generar colores
void color (int rojo, int verde, int azul) {
  analogWrite(pinLedR, rojo);
  analogWrite(pinLedV, verde);
  analogWrite(pinLedA, azul);
}
```

Circuito:



Explicación:

El código es bastante simple, pero hemos agregado unas cuantas cosas extras, como por ejemplo usar una función, esto ya se vio en los primeros temas.

Primero definimos los pines a usar, en este caso no usamos **#define** para cambiar un poco el código, y en lugar del **#define** usamos **variables del tipo entero**, esto se explicó en los temas anteriores, se creó una

variable del tipo global llamada **pausa** para usar un **delay universal**, si se cambia el valor de esta variable cambiará la velocidad de cambio en todo el programa.

En el **setup** se configuran los pines como salida, no pasamos a la explicación del **loop** ya que para eso primero debemos explicar la función **color**.

Se creó una función del tipo **void** ya que no regresará ningún valor y es llamada **color**, esta función recibirá **tres parámetros del tipo entero** que serán valores enteros o variables con un **valor entre 0 y 255**, estos parámetros pasan a la función **analogWrite**, en simples palabras, la función color se encarga de manipular los colores del led RGB.

Ya en el **loop**, mandamos llamar a la función varias veces para poder hacer funcionar el led RGB, hacemos mezcla de colores con los colores básicos, rojo, verde y azul.

Los valores de la función pueden ser manipulados con tres potenciómetros para poder hacer la mezcla de colores, esto se le deja al lector hacerlo.

Con esta última práctica cerramos el tema de actuadores analógicos, con esto ya deben saber cómo usar la función analogWrite y hacer manipulación del PWM, también a unir los temas anteriores entre sí. A continuación veremos control y manejo de cargas, donde aprenderemos a manipular los 110 o 220 volts alternos de la instalación eléctrica residencial, además de manipular más voltaje directo con el uso de transistores y optoacopladores.

Video [tutorial]: *[Curso Arduino Básico - #6] Actuadores Analogicos*

4.4 CONTROL Y MANEJO DE CARGAS

En este tema se enseñará a controlar distintas cargas como la corriente alterna de nuestras residencias, ya sean los 110 o 220 voltios, además de mayores voltajes a 5 volts, en las prácticas que harán se emplearán los temas vistos anteriormente.

Los pines de Arduino tienes sus límites, esto se debe considerar para los motores y otros circuitos, los límites de voltaje y amperaje ya se comentaron anteriormente pero aquí se vuelve a recordar:

- Las salidas de voltaje son de 5 volts.
- La corriente máxima por pin es de 40mA.
- Un total de corriente de todos los pines de 200mA.

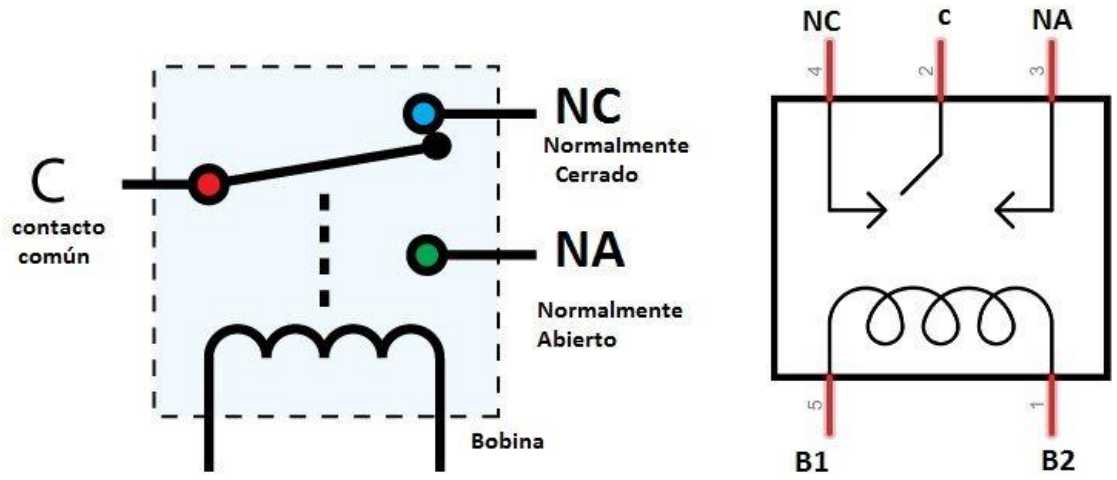
Para poder utilizar dispositivos con un voltaje o corriente mayor podemos usar alguno de los siguientes componentes:

- Relevadores.
- Transistores.
- Optoacopladores.
- Son activados con las salidas del Arduino.

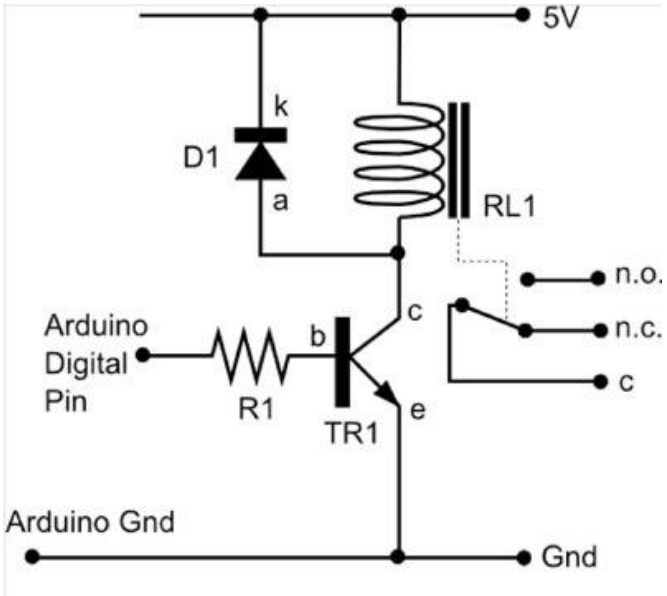
Veamos en que consiste cada uno de estos componentes, se recomienda al lector buscar más información de cada uno de los componentes aquí nombrados para ampliar su conocimiento de estos.

- Relevadores
  - Es un dispositivo electromecánico.
  - Un interruptor mecánico controlado por un circuito electrónico.
  - Una bobina o electroimán acciona uno o varios contactos que sirven como un switch.
  - Se pueden manejar voltajes de 120V/220V AC y también CC.

Símbolos y conexiones del relevador:

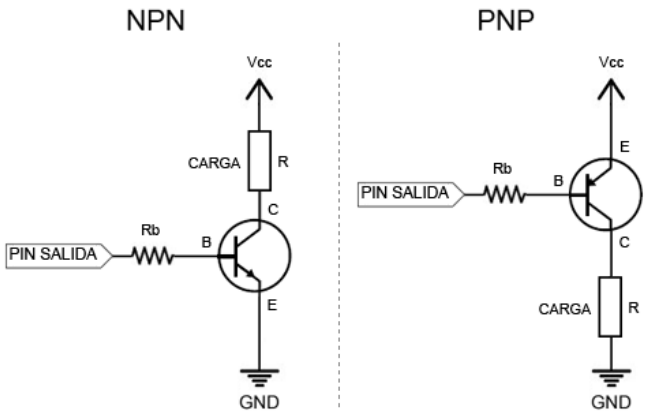


Al meter corriente por la bobina los contactos abiertos se cierran y los cerrados se abren.

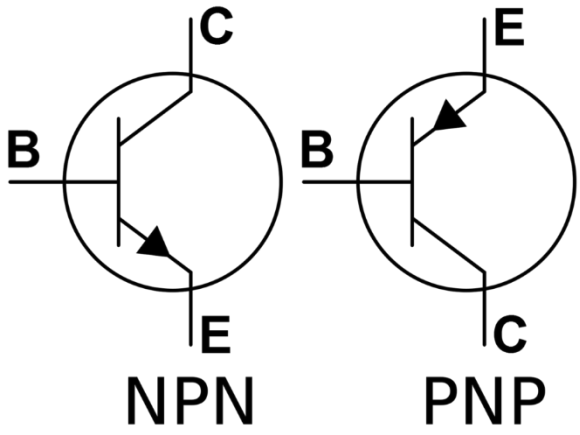


- Transistores
    - Son un dispositivo electrónico semiconductor.
    - Puede ser usado en distintas formas.
        - Amplificador, switch, oscilador, conmutador, etc.
    - Puede ser PNP o NPN y construido con diversas tecnologías y características.
    - Tiene tres conexiones.
        - Base, Colector y Emisor.

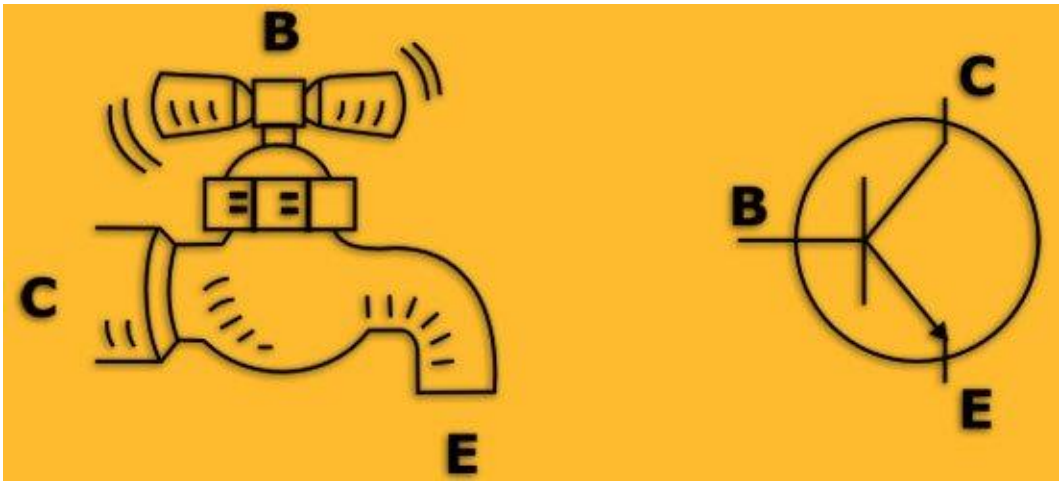
Para una mejor comprensión de estos, imaginémoslo como una llave de agua, la base será la llave para abrir o cerrar así como para controlar si sale más o menos agua, el colector de donde proviene el agua a una alta o baja presión y el emisor la boca de la llave, entre más abrimos la base más agua saldrá en el emisor, dependiendo de la presión del colector saldrá con mayor o menor fuerza, abajo se muestran los símbolos de los transistores y alguna forma de conexión.



Conexión



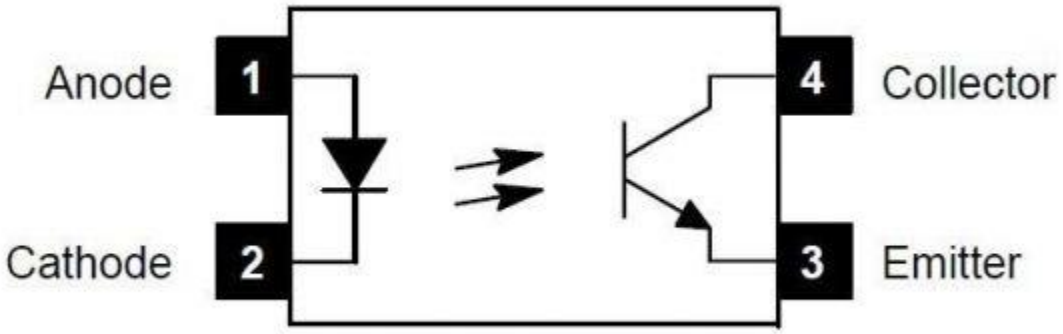
Símbolo

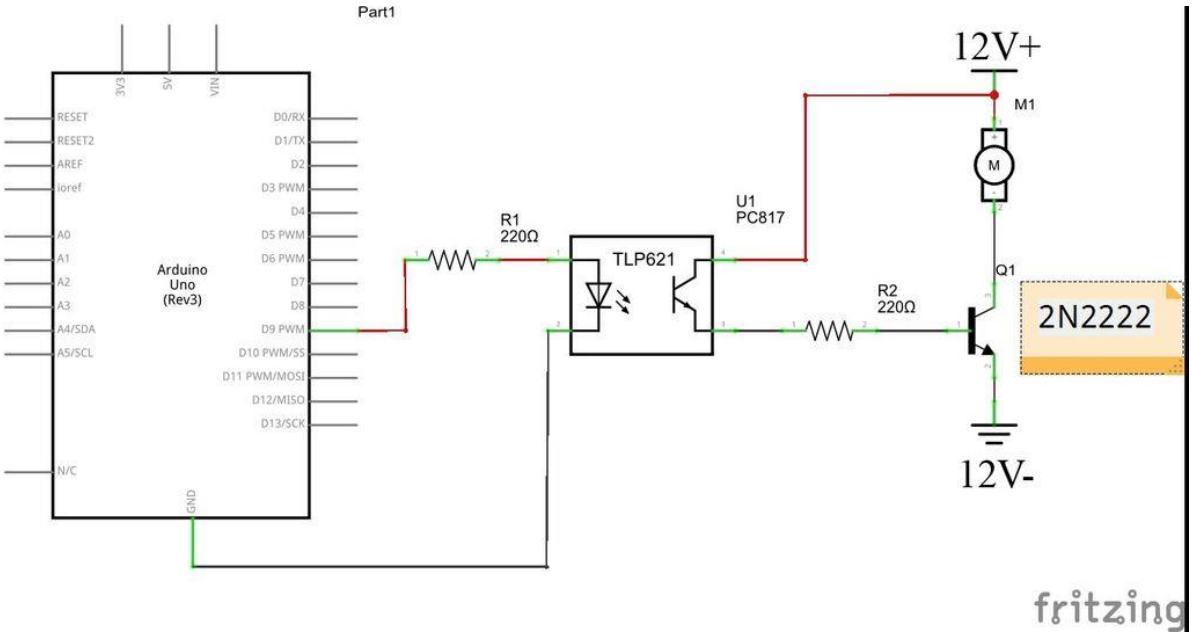


- Optoacopladores
    - Es un circuito integrado o dispositivo semiconductor.
    - Se usa para aislar eléctricamente dos circuitos usando una señal óptica.
    - Tiene un fotoemisor y un fotoreceptor o fototransistor en un solo dispositivo.

Los optoacopladores no solo nos sirven para aislar, sino también para enviar señales y además leer señales digitales de mayor voltaje que 5 volts.

Símbolos y conexiones del optoacoplador:





A continuación, en las prácticas siguientes aprenderemos el uso de los componentes anteriormente mencionados junto a lo ya aprendido en temas anteriores.

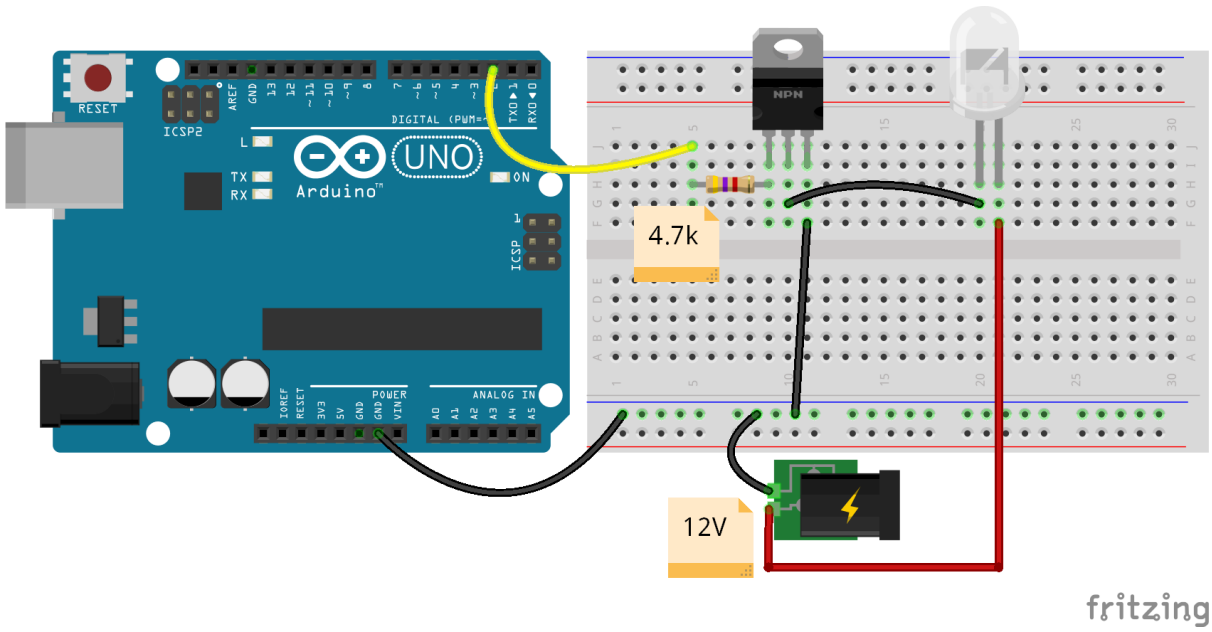
4.4.1 USO DEL TRANSISTOR

Como bien mencionamos, en esta parte se apegará más al manejo de los componentes ya mencionados para que controlen actuadores que requieren más voltaje o amperaje, se emplea lo aprendido en los temas anteriores. En la siguiente práctica se muestra el uso del transistor, abajo se muestra el código, material y explicación de la misma.

MATERIALES
1 ARDUINO
1 Foco de 12V de CC
1 TIP122
1 Resistencia de 4.7K
1 Fuente de 12V
Protoboard
Jumpers

```
#define Trans 2 //Definimos el pin de salida a la base del transistor

void setup() {
  pinMode(Trans, OUTPUT); //Ponemos como salida el pin
}

void loop() {
  //Usamos el metodo Blink con un parametro de 1000ms
  Blink(1000);

}

/**
 * Metodo Blink
 *  @param i -Recibe un entero en ms
 */
void Blink(int i) {
  digitalWrite(Trans, HIGH);
  delay(i);
  digitalWrite(Trans, LOW);
  delay(i);
}
```

Circuito:



Explicación:

El código es muy fácil de entender, se definieron los pines y las variables a utilizar, lo que se hizo fue crear una función de **Blink** para que se prenda y pague el foco mediante el pin 2, pero esta función recibe un parámetro del tiempoque será el valor que tendrá el **delay**, ya en **loop** se llama esta función, y verán como prende y apaga el foco, si editamos el **parámetro** de la función **Blink** verán que aumenta el tiempo en que prende y paga (mayor o menor velocidad).

Ahora veremos lo interesante, como pueden observar estamos manejando voltajes mayores a 5 voltios, en este caso 12 voltios, además de una gran cantidad de amperaje, entre 500mA y 1000mA, esto es gracias al transistor TIP122 del tipo NPN, nosotros enviamos el pulso por el pin 2 del Arduino a la base del transistor, que a su vez está conectada a una resistencia de 4.7k, la función de esta resistencia es limitar la corriente que consume el transistor ya que para la excitación de la base no necesita mucha, al hacer esto la base se excita y abre la llave (como se explicó anteriormente), entonces fluye la corriente del colector al emisor con un voltaje de 12 volts. Como también se pudo observar, cuando usamos una fuente de poder externa, en todos los casos se unen los negativos tanto del Arduino y el de la fuente externa, también llamado GND o tierra.

Con esto podemos observar cómo controlar voltajes mayores y también un amperaje mayor, con esta base ya podemos controlar tiras de leds, motores en una dirección (se verá más adelante), etc. En la siguiente práctica se verá PWM con el transistor.

Una observación es que se puede manejar mayor voltaje y amperaje con los transistores, solo es cuestión de elegir el correcto. ¿Y cómo sabremos que voltaje y amperaje soporta? se preguntarán, pues es muy fácil, basta con leer la hoja de datos del transistor.

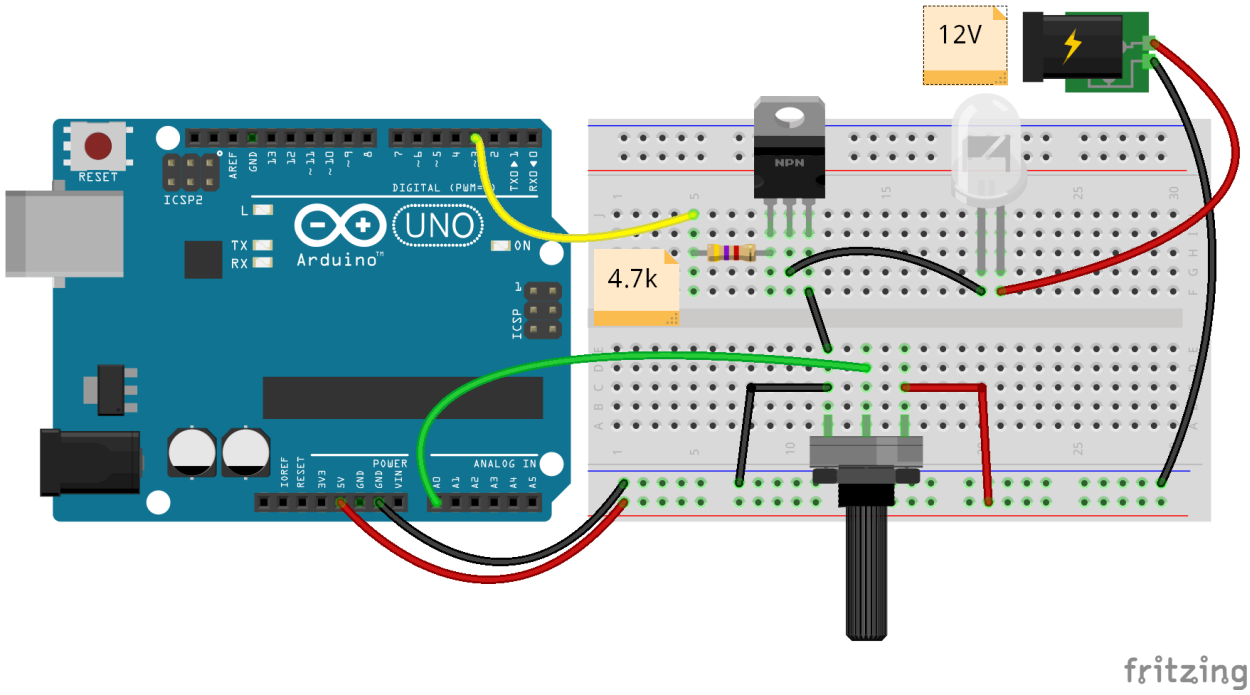
4.4.2 PWM CON TRANSITOR

En esta práctica se verá el control del PWM con el uso del transistor y el foco de 12V, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Foco de 12V de CC
1 TIP122
1 Resistencia de 4.7K
1 Fuente de 12V
1 Potenciometro de 10k
Protoboard
Jumpers

```
#define FOCO 3
#define POT A0

void setup() {

  Serial.begin(9600);
  pinMode(POT,INPUT);
  pinMode(FOCO,OUTPUT);

}

void loop() {

  int valor = analogRead(POT);

  int mapeo = map(valor,0,1023,0,255);

  Serial.print("Valor POT: ");
  Serial.print(valor);
  Serial.print(" | Valor PWM: ");
  Serial.println(mapeo);

  analogWrite(FOCO,mapeo);
  delay(100);

}
```

Circuito:



Explicación:

El código es exactamente igual al de control de PWM con un led y la explicación es la misma, en este caso lo que cambia es la electrónica, como ven usamos nuevamente el transistor para controlar voltajes y corrientes altos, en este caso 12 volts.

Enviamos los pulsos del PWM por el pin 3, esto llega a la base del transistor y dependiendo del pulso del PWM excitará más o menos la base del transistor y eso hará que brille más o menos el foco de 12 voltios, esto es muy interesante porque “cambiamos” la referencia de voltajes, en este caso recordemos que el PWM es de 8 bits así que como máximo tendremos un valor de 255 que en la primera práctica eran 5 volts, pero en esta nueva práctica la referencia cambia, esto implica que 255 son 12 voltios gracias a nuestro transistor, y no solo podemos controlar leds, podemos controlar también sirenas, buzzer, etc.

Bien, ya hemos aprendido como usar los transistores para poder controlar actuadores que requieren un mayor voltaje y amperaje, recuerden que esto es la base del aprendizaje, se le deja al lector que experimente con otros componentes.

En la siguiente práctica aprenderemos a usar los relevadores con Arduino y poder controlar las instalaciones eléctricas.

4.4.3 USO DEL RELEVADOR CON ARDUINO

En esta nueva práctica se enseñará a usar un relevador con Arduino y controlar los aparatos eléctricos o voltajes mayores, esto puede ser usado en la domótica para poder prender la luz, televisor, etc.

Se recomienda el uso de un módulo relay para esta práctica, si usted no cuenta con uno le recomendamos lo adquiera o bien puede hacerlo de manera casera como se enseña en el siguiente video tutorial.

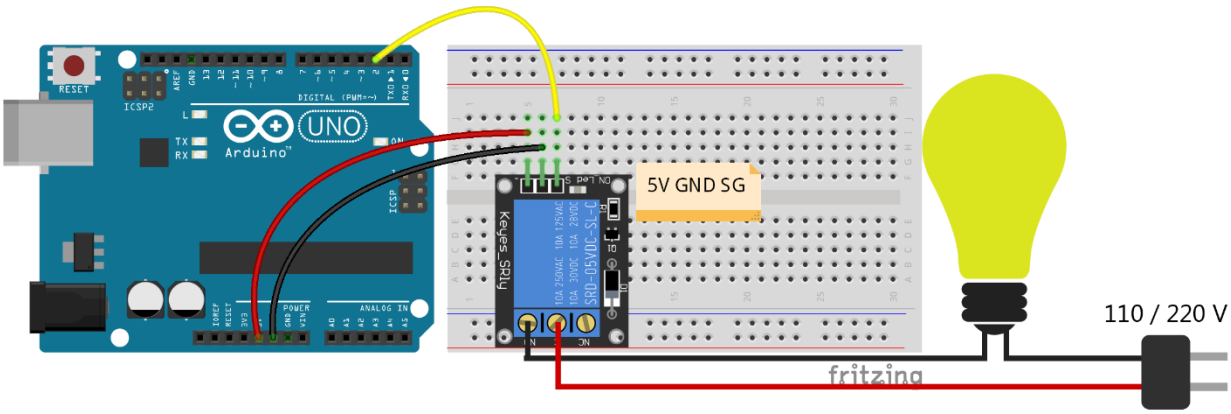
Video [tutorial]: *[Tutorial Electronica] Circuitos Utiles – Modulo Relay 5V para Microcontroladores*

Abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Foco con extensión de CA
1 Módulo Relay de 5V
1 Fuente de 5V
Protoboard
Jumpers

```
#define FOCO 2

void setup() {
    pinMode(FOCO, OUTPUT);
}

void loop() {
  //Metodo Blink
  Blink(1000);
}

/**
 * Metodo que ejecuta un Blink usando un relevador para
 * poder controlar un foco a 120V AC
 * @param i - Recibe un entero que es el tiempo de retardo en ms
 */
void Blink(int i) {
  digitalWrite(FOCO, HIGH);
  delay(i);
  digitalWrite(FOCO, LOW);
  delay(i);
}
```

Circuito:



Explicación:

Como pueden observar el código es bastante simple y no se entrará en explicación ya que se vio en temas anteriores, lo interesante es la parte electrónica, es recomendable que hayan visto el video sugerido de crear su módulo relay, en este caso como solo es un relevador podemos usar la alimentación del Arduino pero se recomienda usar alimentación externa de 5 volts.

Como pueden ver en el código se envía un **pulso** por el **pin 2,** esto llega a la base del transistor del tipo npn excitándola, lo que hace que fluya mayor corriente y se magnetice la bobina del relevador, eso atrae el contacto interno y hace que pase el voltaje del común al normalmente abierto, lo cual hace que se pueda prender nuestro foco u otros aparatos electrónicos, usted puede hacer pruebas con radios, televisores, etc.

En este caso no funciona el PWM ya que es necesario excitar la bobina por completo, pero si usted quiere controlar el PWM para iluminación pude investigar acerca de los MOC y TRIAC.

Con esto usted ya sabe cómo controlar voltaje de 110/220v, incluso ya puede hacer un estrobo uniendo esta práctica con las primeras que se vieron en actuadores digitales, además junto con lo que se verá más adelante que será el tema de la comunicación serial, usted podrá controlar los aparatos y luces de su casa o propiedad con su teléfono móvil.

4.4.4 USO DEL OPTOACOPLADOR

En este tema veremos cómo usar el optoacoplador para activar actuadores y leer sensores digitales que envían pulsos de más de 5v, por ejemplo de 12 voltios hasta 24 voltios o más.

El funcionamiento del optoacoplador es bastante simple pues dentro contiene un diodo led emisor y un fotoreceptor. Cuando se enciende el led, el fototransistor es excitado y funciona dependiendo de cómo se conecte, como pull-up o pull-down.

Las siguientes dos prácticas son muy simples, en la primera prendemos un foco de 12v y en la segunda leeremos ese voltaje.

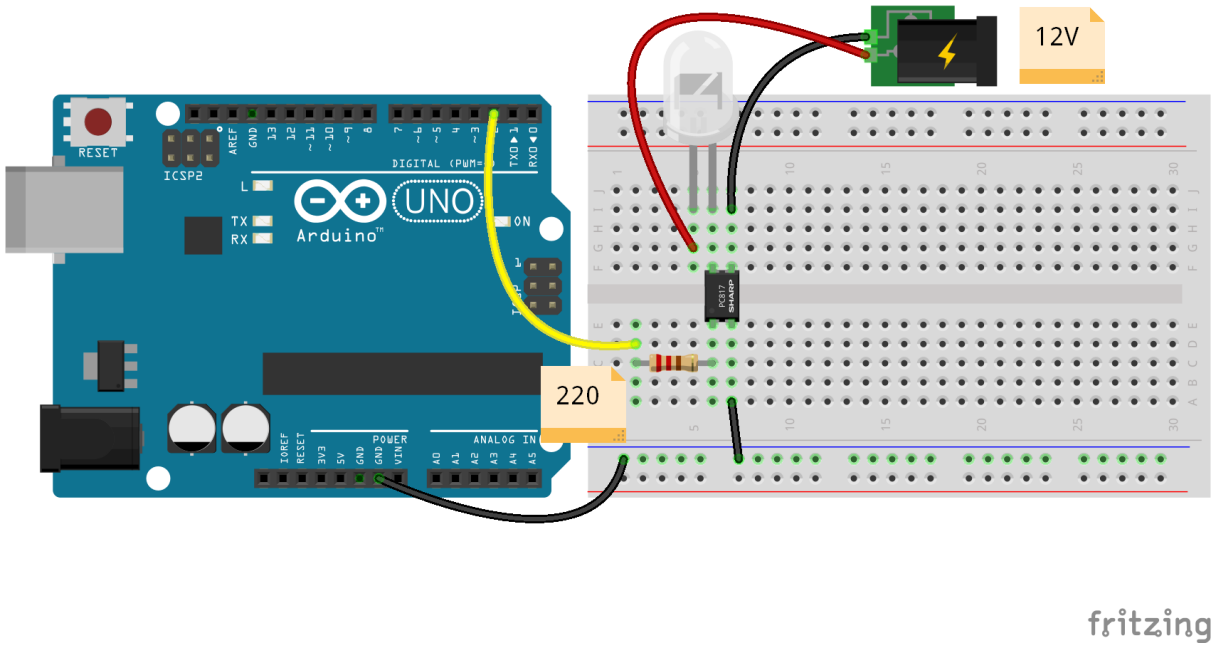
4.4.4.1 OPTOACOPLADOR COMO ACTUADOR

En esta práctica se pretende hacer funcionar el optoacoplador para que prenda el foco de 12v, antes que nada el optoacoplador que se usará es el PC817, así que el lector debe de leer la hoja de datos del optoacoplador para ver sus características y su modo de conexión, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Foco de 12V
1 PC817
1 Resistencia de 220 Ohms
Fuente de 12V
Protoboard
Jumpers

```
//Definimos el pin del emisor del Optoacoplador
#define Opto 2

void setup() {

  //Definimos como salida
  pinMode(Opto, OUTPUT);

}

void loop() {

  //Hacemos una especie de Blink durante 2000ms
  digitalWrite(Opto, HIGH);
  delay(2000);
  digitalWrite(Opto, LOW);
  delay(2000);

}
```

Circuito:



Explicación:

El código es bastante simple y a esta altura usted ya debe de ser capaz de comprenderlo, pues ya se ha visto en anteriores prácticas, además de las funciones que se usan ya se han explicado a fondo, lo interesante como ya se ha mencionado antes es en la parte electrónica.

La función es simple, se envían pulsos digitales de 2000 milisegundos por el **pin dos**, esto al pin uno del optoacoplador pues es ánodo de éste, como ya se mencionó al principio de este tema, el optoacoplador está compuesto por un led emisor y un fototransistor, lo que hacemos es prender ese led, al prender el led se excita la base del fototransistor que es un transistor tipo npn, así que como ya se vio en la parte de transistores, fluye la corriente del **emisor** (pin tres) del optoacoplador al **pin cuatro** que es el colector y así de simple.

Como se pudo observar no se tuvo que poner tierra común, esta es una ventaja del optoacoplador ya que "hace una división" de su funcionamiento, por eso se usa para proyección donde se generan picos y así no dañar nuestro microcontrolador, Arduino, etc.

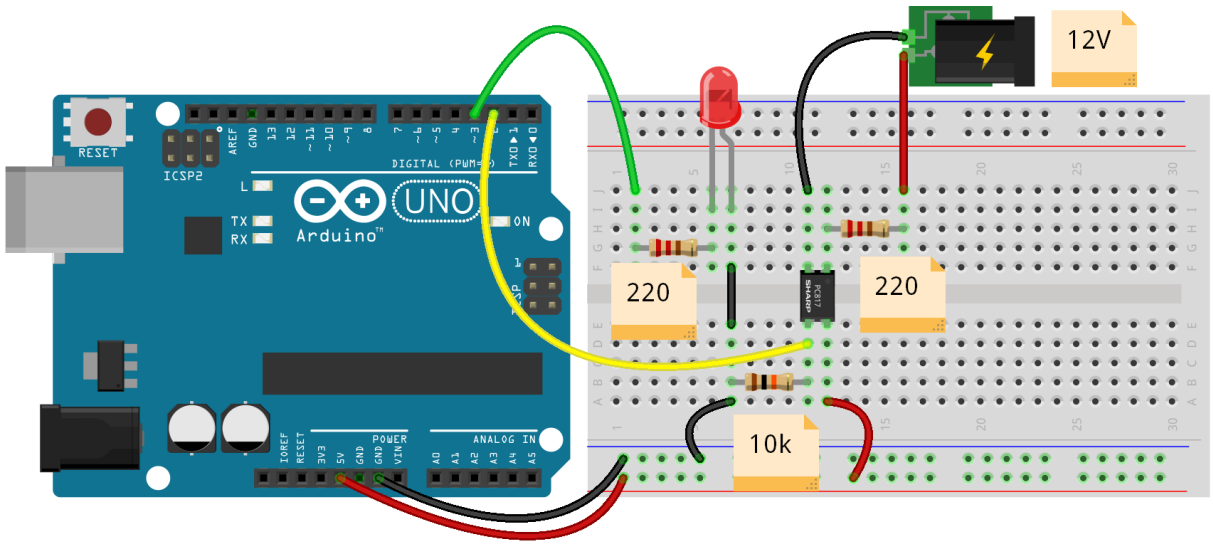
4.4.4.2 LEER SEÑALES DE MÁS DE 5V

En esta y ultima práctica del tema de control de cargas veremos cómo leer señales digitales de más de 5 voltios, hay sensores industriales que envían señales de 12 o 24 voltios, para leer estas señales nos ayudará nuestro optoacoplador, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Led de 5mm
1 PC817
2 Resistencia de 220 Ohms
1 Resistencia de 10k
Fuente de 12V
Protoboard
Jumpers

```
//Definimos los pines a usar
#define Sensor 2
#define LED 3

void setup() {

  //Ponemos el sensor como entrada y el led como salida
  pinMode(Sensor, INPUT);
  pinMode(LED, OUTPUT);

}

void loop() {

  //Metodo prende
  prende();

}

/**
 * Este metodo lee el estado del sensor en una variable
 * axiliar, si esa variable es 1 o HIGH prende el led
 * en otro caso se mantiene apagado.
 * Leemos 12V
 */
void prende() {

  int aux = digitalRead(Sensor);
  if (aux == HIGH) {
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }

}
```

Circuito:



Explicación:

Como se puede observar se definen los pines a usar, uno para leer la señal y otro para enviar un pulso al led, en el **loop** se llama el método **prende**, este método es definido más abajo, el cual consiste en leer el pulso enviado al **pin 2** del Arduino y con uso de la sentencia de control **if**, se hace un comparación con el valor almacenado en la variable **aux** sobre si este valor es alto, o sea si el sensor envía una señal prende el led, en otro caso el led permanece apagado.

Lo interesante está en la parte electrónica, como se puede observar en el diagrama de conexión, simulamos un sensor de 12 voltios con la fuente de 12 voltios, cuando se prende o conecta la fuente hace que prenda el **led emisor** interno del **pc817**, esto excita la base del **fototransistor**, la conexión del fototransistor está en forma de **pull-down** con los 5 voltios del Arduino (si usan más es necesario una fuente externa), esto quiere decir que cuando el led emita o prenda se enviará un pulso alto de 5 voltios al **pin 2,** al ser detectado en la parte de la función prende y se ejecutará la condición antes mencionada.

Y es así como podemos leer señales digitales de mayor voltaje con nuestro Arduino, como siempre se menciona esto es la base del aprendizaje, se le deja al lector que experimente con la debida precaución de no quemar la placa Arduino, en el siguiente tema se verá el uso y control de motores.

Video [tutorial]: *[Curso Arduino Básico - #7] Control y manejo de cargas*

4.5 USO Y MANEJO DE MOTORES

En este tema se enseña el uso y manejo de motores, se recuerda que se enseñará la base del uso de estos, el uso que se le dará se deja al lector, se mostrará cómo usar un motor con un transistor, el uso del puente h, del servomotor y motor a pasos.

Servomotor con Arduino

- Se pueden controlar hasta 19 servos usando los pines analógicos.
- Se usará la librería incluida en Arduino llamada Servo.
    - Para usarla debemos escribir arriba del sketch: #include <Servo.h>
    - Declararemos nuestro servomotor con: Servo [nombre del servo];
    - En setup declararemos el pin del servo: [Nombre del servo].attach([pin]);
    - Para escribir en nuestro servo usamos: [Nombre del servo].write([angulo]);

Podemos mover un servo sin librería con esta función:

```
void moverServo(int pin, int angulo) {
  float pausa;
  pausa = angulo * 2000.0/180.0+500;
  digitalWrite(pin,HIGH);
  delayMicroseconds(pausa);
  digitalWrite(pin,LOW);
  delayMicroseconds(25000-pausa);
}
```

Para el ejemplo que veremos usaremos un servomotor SG90, ya que puede ser alimentado con los 5 voltios del Arduino sin ningún problema, si se usan más de uno o un servomotor que requiere más amperaje que el SG90, es necesario que se use una fuente externa que entregue el amperaje necesario para los servomotores.

Motores DC con Arduino

El uso de motores de DC requieren una forma de controlarlos, lo cual puede ser un transistor como vimos en control de cargas, el transistor puede manejar altos voltajes y corrientes, también si quiere un control un poco más preciso es necesario el uso de un puente H, existen dos puentes H muy usados; el L293D y el L298N, el puente H a usar dependerá del voltaje y consumo de corriente de sus motores, en este caso usaremos un motor reductor de 6 voltios y 200mA pico, que es el común amarillo muy usado, a continuación se verá el puente H ya que el uso del transistor ya se vio en temas anteriores.

Puente H L293D

Se recomienda leer como siempre la hoja de datos de este componente, pues allí contiene la información más importante, el voltaje y amperaje que soporta además de cómo usarlo, aquí se hará una pequeña descripción.

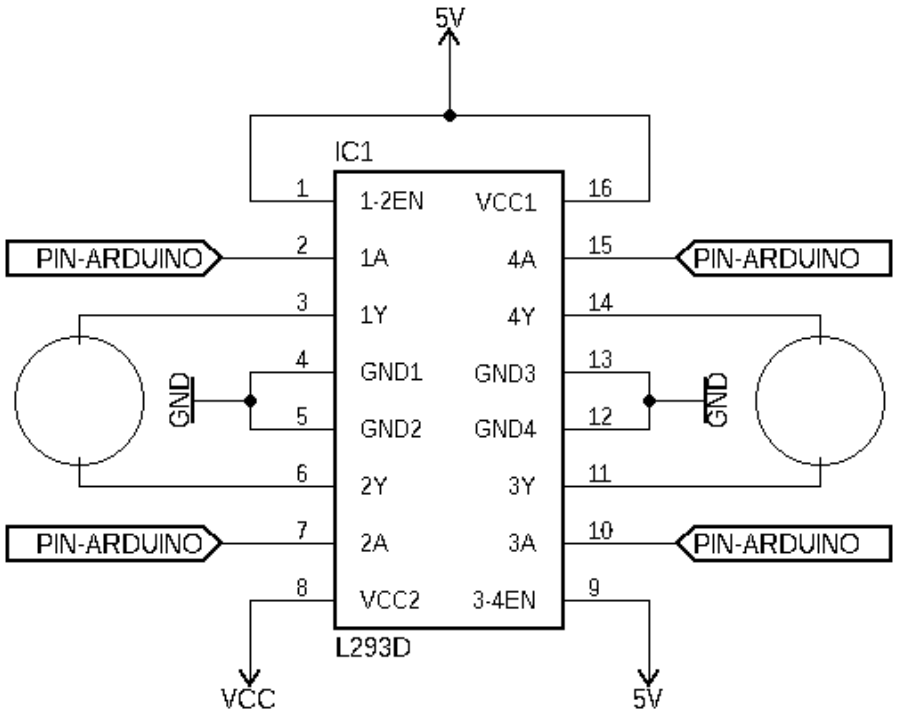
El puente H L293D soporta 400mA por canal, o sea 800mA en total y 1000mA pico, la lógica del puente H es de 5 voltios, este voltaje se conecta en el pin 1, 16 y 9. La alimentación de los motores o sea el positivo se conecta en el pin 9, los pines 4, 5, 12 y 13 van conectados a GND, recuerde que GND es común así que se conectan tanto el GND del Arduino como el de la alimentación exterior, en los pines 3 y 6 se conecta un motor, en los pines 11 y 14 se conecta un segundo motor y finalmente en los pines 2, 7 y también los pines 10 y 15 se conectan al microcontrolador, esto es para mover el motor, más adelante en la práctica se verá más a fondo.

Es muy importante también considerar la tabla de verdad del puente H la cual se pone más abajo con un diagrama del L293D y su forma de conexión.

Tabla de verdad:

1A – 4A	2A – 3A	FUNCION
L	H	Gira hacia adelante
H	L	Gira hacia atrás

Diagrama de conexión:

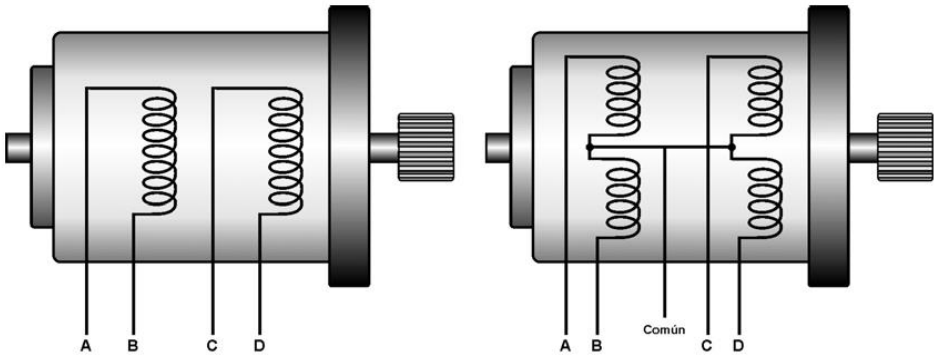


En el diagrama, en la parte que dice VCC se refiere al positivo de la alimentación de los motores.

Motores paso a paso

Los motores a paso son muy usados cuando se requiere de movimientos muy precisos como por ejemplo una CNC o una impresora 3D, existen dos tipos de motores a pasos; los motores bipolares y los unipolares, las diferencias de estos es que difieren en su embobinado interior, si requiere más información puede encontrar en la red una gran cantidad de artículos sobre estos motores, en este caso se verá el ejemplo con un motor bipolar el cual puede controlarse con un puente H, los motores a pasos como lo dice su nombre se mueven a pasos, existen diferentes tipos, en este ejemplo se verá **FULL STEP**, los demás motores a pasos se le dejan investigar al lector, abajo se muestra el motor a pasos bipolar y unipolar así como la forma de conexión para cada uno.

Motor bipolar **Motor unipolar**



Conexión motor bipolar:

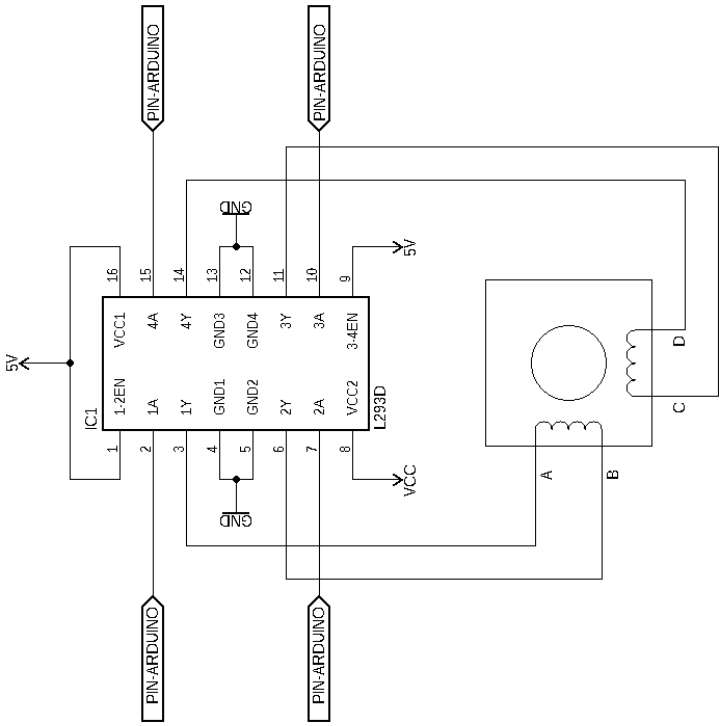


Tabla de verdad FULL STEP:

A	B	C	D
HIGH	LOW	HIGH	LOW
LOW	HIGH	HIGH	LOW
LOW	HIGH	LOW	HIGH
HIGH	LOW	LOW	HIGH
1A	2A	3A	4A
FULL STEP MOTOR BIPOLAR			

En las siguientes prácticas queda más claro el uso de los motores y toda la información ya dicha, se les recuerda leer las hojas de datos de cada componente a usar.

4.5.1 USO DEL SERVOMOTOR

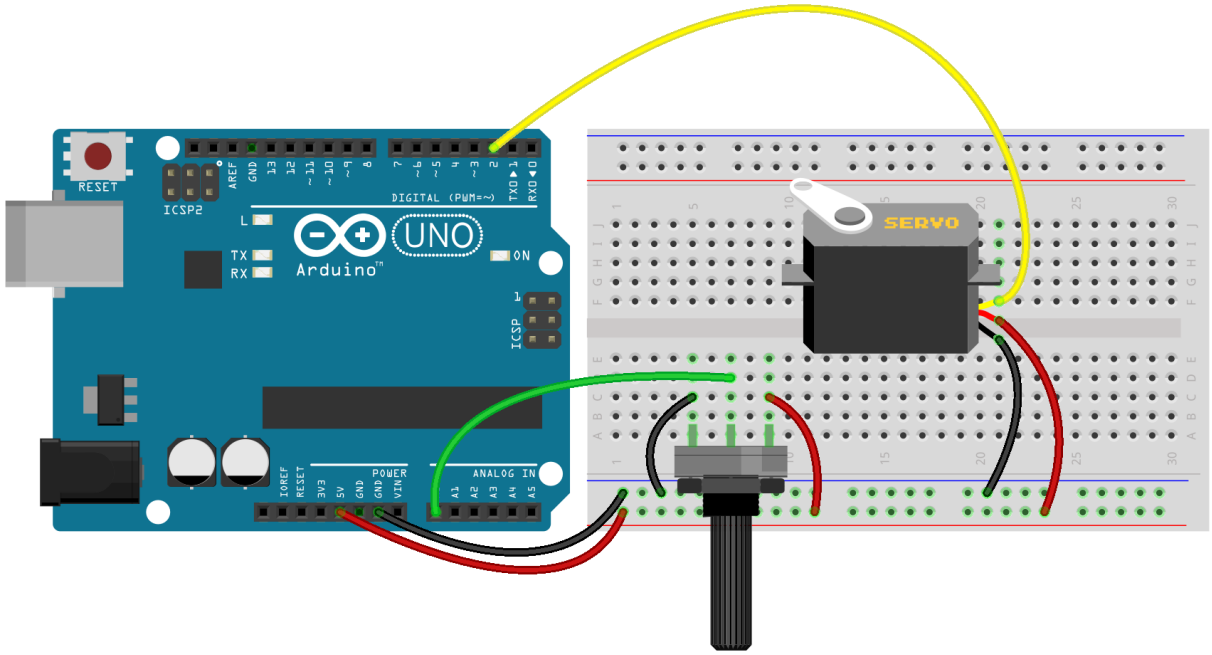
En esta práctica se enseña a usar el servomotor con la librería que ya incluye el software de Arduino, aquí se usa lo aprendido anteriormente, por ese motivo solo se explicará el código importante, el servomotor que se usará será un SG90, este servomotor es de 180°, además de que esto es la base para poder crear proyectos más avanzados uniendo lo ya visto y los temas por ver, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Motor servo SG90
1 Potenciómetro de 10k
Protoboard
Jumpers

```
//Incluimos la libreria servo
#include <Servo.h>

//Definimos pines a usar
#define POT A0

//Definimos nuestro servo
Servo ServoUNO;

void setup() {
  Serial.begin(9600);
  pinMode(POT,INPUT);
```

```
  ServoUNO.attach(2);
}

void loop() {
  int POT_R = analogRead(POT);

  int aux = map(POT_R,0,1023,0,180);

  ServoUNO.write(aux);
  delay(10);

  Serial.print("Valor POT: ");
  Serial.print(POT_R);
  Serial.print(" | Angulo Servo: ");
  Serial.println(aux);

}
```

Circuito:



Explicación:

Como pueden observar el código se inicia llamando la librería del servo usando **#include <Servo.h>**, después se definen los pines a usar; en este caso el potenciómetro y el servo al que se le asigna el nombre de **ServoUNO**. En el **setup** se configuran los pines con la función de la librería **attach** enviándola al nombre del servo, se declara en qué pin está conectado el servo; en este caso en el **pin 2** como se ve en el diagrama.

En la parte del **loop** puede observar que se leen los datos del potenciómetro y se almacenan en una variable, después usando la función **map** vista anteriormente se transforman los valores del potenciómetro a valores aceptados por el servomotor, en este caso el servomotor es de 0 a 180°, es por eso que, gracias a la función **map** trasformamos a los valores de 0 a 180.

A continuación, usando la función **write** de la librería **Servo** y enviándola al **nombre del servomotor**, le entregamos como parámetro la variable **aux**, ya que en esa variable se almacenarán los números de 0 a 180, se asigna un delay para que el servomotor no se mueva muy rápido, cuando intente mover el potenciómetro usted notará que el servomotor también se mueve desde 0 a 180°.

Usando el monitor serial podemos ver los valores del potenciómetro y a qué grado se convierte cada valor, como puede ver hemos unido lo visto en sensores analógicos para mover un servomotor, también puede unir lo visto en sensores digitales para mover el servomotor, esto queda como investigación al lector.

Más delante en el Capítulo 5, se verá un proyecto avanzado con servos, para comprenderlo será necesario que entienda este tema.

4.5.2 MOTOR DC CON TRANSISTOR

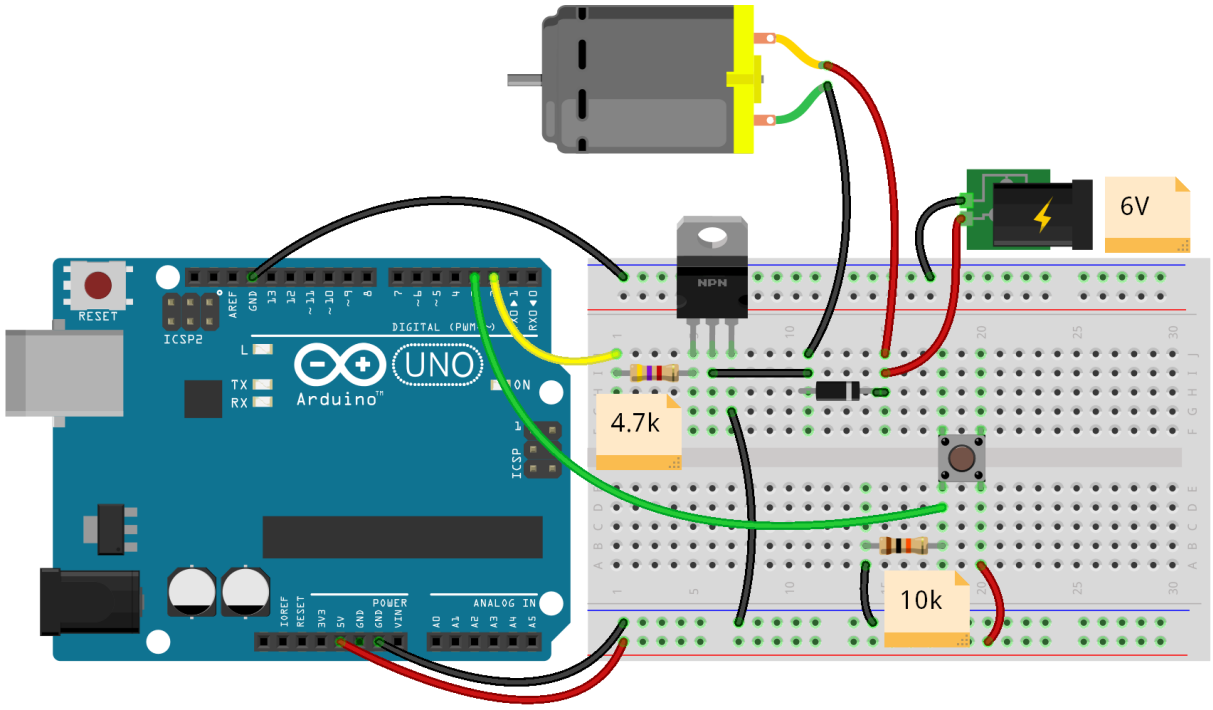
En esta práctica emplearemos un motor DC con un transistor, con el cual también podemos controlar el PWM, pero solo puede ir en una dirección dependiendo de la "polaridad" de los cables del motor, no se entrará en ninguna explicación del código pues ya se vio en control de cargas, solo se explicará un detalle de la parte electrónica, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Motor reductor de 6V
1 Fuente de 6V pueden ser 4 pilas AA
1 Tip122
1 Resistencia de 4.7k
1 Resistencia de 10k
1 Pulsador de 4 pines
1 Diodo 1N4004
Protoboard
Jumpers

```
//Definimos pines a usar
#define Motor 2
#define Pulsador 3

//Variables a usar
int Push;
```

```
void setup() {
  //Configuramos los pines como entrada o salida
  pinMode(Motor, OUTPUT);
  pinMode(Pulsador, INPUT);
  digitalWrite(Motor,LOW);
}

void loop() {
  Push = digitalRead(Pulsador);

  if (Push == HIGH) {
    digitalWrite(Motor, HIGH);
  } else {
    digitalWrite(Motor, LOW);
  }

}
```

Circuito:



Explicación:

El código es bastante simple, debido a que ya se vio todo esto en temas anteriores no se entrará en detalles, lo único que tiene extra es la lectura de un pulsador con una sentencia de control if, para que prenda o no el motor.

El cambio está en la parte electrónica, lo único diferente a lo ya explicado es que usamos un diodo 1N4004 entre la salida del colector del TIP122 que está unida a un pin del motor y el voltaje de alimentación que está unido al otro pin del motor, esto se hace porque el motor genera un campo electromagnético para que pueda girar. Cuando se deja de alimentar o se le quita el voltaje, este campo se convierte en voltaje y corriente que deben de fluir hacia algún lado, por eso el uso del diodo, gracias a este componente ese flujo no va a ningún lado y no estropeará nuestro Arduino.

En la siguiente práctica se muestra también como controlar un motor pero con un puente H.

4.5.3 MOTOR CON PUENTE H

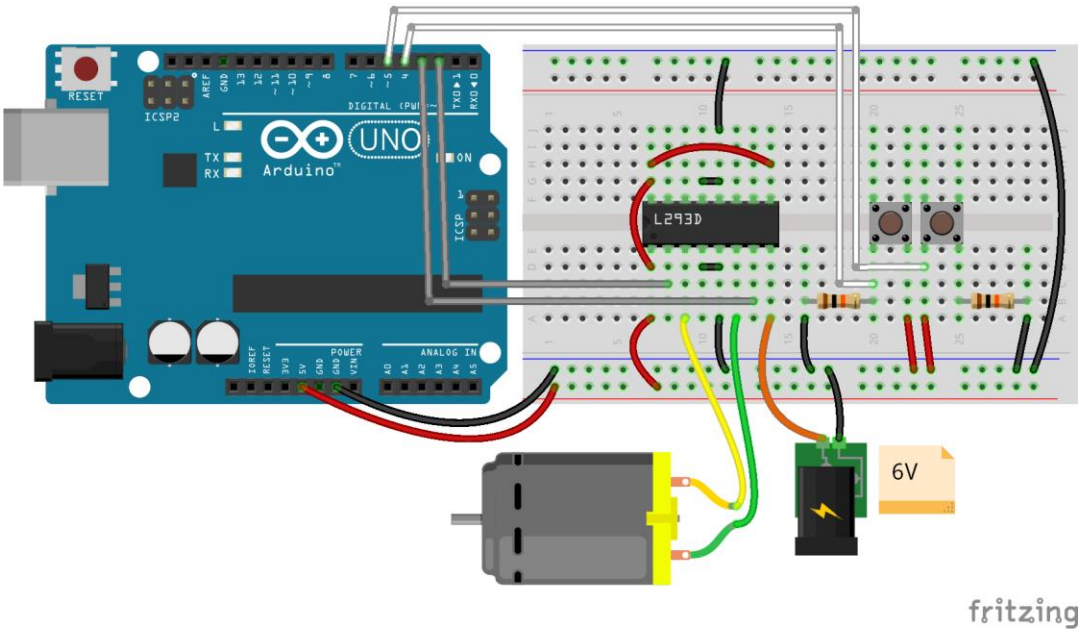
En esta práctica se enseñará el uso y control de un motor DC con un puente H, este puente H nos dará un poco de más control sobre el motor ya que podremos cambiarlo de dirección, lo cual se verá en la práctica además de modificar la velocidad del motor (PWM) pero esto se le deja al lector investigarlo. Abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Motor reductor de 6V
1 Fuente de 6V pueden ser 4 pilas AA
1 L293D
2 Resistencia de 10k
2 Pulsador de 4 pines
Protoboard
Jumpers

```
//Definimos los pines
#define MotorUno_1 2
#define MotorUno_2 3

#define pulsador_uno 4
#define pulsador_dos 5

int puldador_UR;
int pulsador_DR;

void setup() {
```

```
  //Configuramos los pines como salida o entrada
  pinMode(MotorUno_1, OUTPUT);
  pinMode(MotorUno_2, OUTPUT);
  pinMode(pulsador_uno, INPUT);
  pinMode(pulsador_dos, INPUT);
  //Inicamos el motor en apagado
  digitalWrite(MotorUno_1, LOW);
  digitalWrite(MotorUno_2, LOW);
}

void loop() {
  puldador_UR = digitalRead(pulsador_uno);
  pulsador_DR = digitalRead(pulsador_dos);

  if (puldador_UR == HIGH) {
    digitalWrite(MotorUno_1, HIGH);
    digitalWrite(MotorUno_2, LOW);
  }
  if (pulsador_DR == HIGH) {
    digitalWrite(MotorUno_1, LOW);
    digitalWrite(MotorUno_2, HIGH);
  }
  if(puldador_UR == LOW && pulsador_DR = LOW){
    digitalWrite(MotorUno_1, LOW);
    digitalWrite(MotorUno_2, LOW);
  }
}
```

Circuito:



Explicación:

El código es bastante fácil de entender si ha prestado atención y aprendido los temas anteriores. Se definen los pines que se usarán, utilizaremos dos pulsadores; cada uno para cambiar el movimiento del motor. En el **setup** se inicializan los pines, además de enviar un pulso bajo a los **pines 2 y 3** para que no haya ninguna interferencia de ruido externo. En el **loop** tenemos tres sentencias **if**; en la primera si presionamos el primer pulsador el motor se moverá a un lado, eso lo hacemos usando el **digitalWrite** además aplicando la **tabla de verdad** del L293D vista anteriormente, en el segundo **if** cambiamos la dirección del motor, simplemente cambiando el valor del pulso que se envía al motor por los **pines 2 y 3** como usted lo puede notar, y eso se ejecutará cuando presionemos el segundo pulsador.

Lo interesante está en el tercer **if** pues tenemos una operación **"Y"** con sintaxis en programación **"&&"**, esto implica que deben ser verdaderas las condiciones antes y después del **"&&"** para que se ejecuten las sentencias que están dentro del **if**, se recomienda al lector que investigue la tabla de verdad de la operación **conjunción**, bueno, esto quiere decir que ambos pulsadores no deben presionarse para que el motor esté apagado, o sea deben enviar un LOW, si una de estas condiciones no se cumple no entra al if.

Y como pueden ver en el diagrama, es muy sencilla la implementación del L293D para controlar motores, el manejo del PWM se le deja al lector investigarlo.

4.5.4 USO DEL MOTOR A PASOS BIPOLAR

En esta práctica se enseña el uso de un motor a pasos bipolar en modo tradicional ya que Arduino ya cuenta con una librería para controlar estos motores, el uso de la librería se le deja de investigación al lector.

El motor a pasos que usted debe de usar no debe de superar los 800mA y el voltaje máximo debe de ser de 36 voltios, pues es lo que soporta el L293D. Se recomienda el motor Nema 17 de 2.4Kg/cm de 200 pasos, ya que de acuerdo a su hoja de datos consume 0.33ª, se recomienda que el lector lea la hoja de datos del motor.

En la práctica se mostrará como girar el motor en modo horario y anti horario, abajo se muestra el código, material y explicación de la práctica.

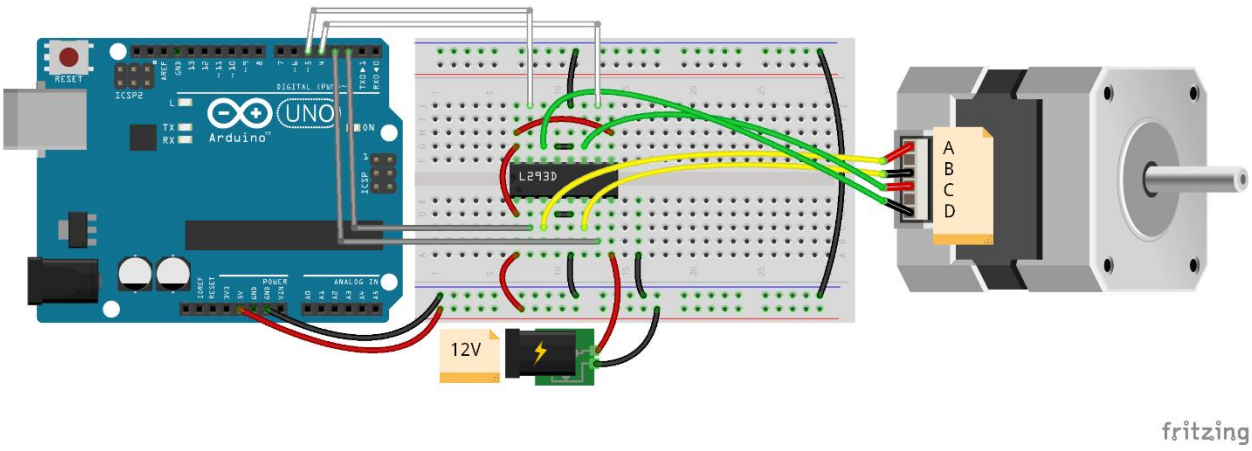
MATERIALES
1 ARDUINO
1 Motor a pasos bipolar de 12v
1 Fuente de 12v
1 L293D
Protoboard
Jumpers

```
//Definimos los pines a usar
#define BobinaUA 2
#define BobinaUB 3
#define BobinaDC 4
```

```
#define BobinaDD 5

void setup() {
  //Inicalizamos los pines
  pinMode(BobinaUA, OUTPUT);
  pinMode(BobinaUB, OUTPUT);
  pinMode(BobinaDC, OUTPUT);
  pinMode(BobinaDD, OUTPUT);
}

void loop() {
  for(int i = 0; i < 51; i++){
    giroHorario(5);
  }

  delay(1000);

  for(int i = 0; i < 51; i++){
    giroAntiHorario(5);
  }
}

//Metodos para que gire el motor

void giroHorario(int tiempo){
  digitalWrite(BobinaUA, HIGH);
  digitalWrite(BobinaUB, LOW);
  digitalWrite(BobinaDC, HIGH);
  digitalWrite(BobinaDD, LOW);
  delay(tiempo);
  digitalWrite(BobinaUA, LOW);
  digitalWrite(BobinaUB, HIGH);
  digitalWrite(BobinaDC, HIGH);
  digitalWrite(BobinaDD, LOW);
  delay(tiempo);
  digitalWrite(BobinaUA, LOW);
  digitalWrite(BobinaUB, HIGH);
  digitalWrite(BobinaDC, LOW);
  digitalWrite(BobinaDD, HIGH);
  delay(tiempo);
  digitalWrite(BobinaUA, HIGH);
  digitalWrite(BobinaUB, LOW);
  digitalWrite(BobinaDC, LOW);
  digitalWrite(BobinaDD, HIGH);
  delay(tiempo);
}
```

```
void giroAntiHorario(int tiempo){
  digitalWrite(BobinaUA, HIGH);
  digitalWrite(BobinaUB, LOW);
  digitalWrite(BobinaDC, LOW);
  digitalWrite(BobinaDD, HIGH);
  delay(tiempo);
  digitalWrite(BobinaUA, LOW);
  digitalWrite(BobinaUB, HIGH);
  digitalWrite(BobinaDC, LOW);
  digitalWrite(BobinaDD, HIGH);
  delay(tiempo);
  digitalWrite(BobinaUA, LOW);
  digitalWrite(BobinaUB, HIGH);
  digitalWrite(BobinaDC, HIGH);
  digitalWrite(BobinaDD, LOW);
  delay(tiempo);
  digitalWrite(BobinaUA, HIGH);
  digitalWrite(BobinaUB, LOW);
  digitalWrite(BobinaDC, HIGH);
  digitalWrite(BobinaDD, LOW);
  delay(tiempo);
}
```

Circuito:



Explicación:

En el código se definen los pines a usar que serán los que manden la señal al L293D y este a su vez a las bobinas del motor bipolar. En el **setup** se inicializan los pines, ahora nos saltaremos fuera del **loop** en donde hemos creado dos funciones del tipo **void**; una llamada **giroHorario** y otra **giroAntiHorario**, esta función **recibe un parámetro del tipo entero** que será la **pausa** entre cada transición de la tabla de verdad, esto se muestra al principio de este capítulo.

La segunda función es similar a la primera, solo cambia el sentido de la tabla de verdad, que va de abajo hacia arriba para que gire en sentido contrario.

Ya en el **loop**, vemos que tenemos la sentencia de control **for** que va desde **0** a **menor de 51**, ahora se preguntarán -¿por qué esto?, como el motor es de **200 pasos** y además usamos la tabla de verdad **FULL STEP** vemos que en esta tabla tiene **4 transiciones**, entonces para que dé un giro completo de 360°, como el motor es de 1.8° de giro, es necesario repetir la secuencia de los pasos **50 veces** que multiplicado por **4** pulsos tendremos los **200 pasos**, luego, si cambiamos el **for** en la condición de **menor a 51** por **menor a 26** girará 180°.

Recuerden que el **for** se inicia en **0** y para que sean las **50** veces tenemos que poner **menor a 51**, y es así como controlamos un motor a pasos en forma tradicional, se le recomienda al lector que juegue con el código para entenderlo mejor.

Con esta última práctica se da por visto el tema de control y manejo de motores.

Video [tutorial]: *[Curso Arduino Básico - #8] Uso y manejo de motores*

4.6 COMUNICACIÓN CON Y DESDE ARDUINO

Este tema será un poco corto pues se verá la comunicación serial, en prácticas anteriores se vio un poco el uso de esta comunicación para mostrar datos en el monitor serial, en una de las prácticas usaremos un módulo bluetooth hc06, también puede usar un hc05, aunque en dado caso solo lo usaremos en modo esclavo -¿qué quiere decir esto?, que enviaremos datos desde nuestro teléfono móvil al Arduino.

Comunicación con Arduino

- Todos los Arduinos tienen al menos un puerto serial, UART o USART.
- Usa los pines 0(RX) y 1(TX) y/0 el puerto USB por default.
- Nos permite comunicarnos con el Arduino.
- Lo podemos hacer por el conector USB.
- Tiene un buffer de 128 bytes.

Nuestro Arduino puede funcionar conectado a una computadora o independientemente, puede comunicarse con la computadora a la que está conectado, también a través del puerto serial con otros equipos o usando alguna shield de comunicación.

Para comunicarnos con Arduino usaremos el puerto serie como ya se mencionó, Arduino ya tiene una librería para esta comunicación que se llama **Serial**, incorpora varias funciones pero las esenciales y que se usarán se nombran a continuación:

- Serial.begin([baudios]); - para iniciar la comunicación, recibe un parámetro del tipo entero que será la velocidad de comunicación, todo dependerá de sus módulos, en general son 9600.
- Serial.available(); - para saber si hay datos en el buffer que leer, si no hay datos envía un cero.
- Serial.read(); - para leer los datos de entrada.
- Serial.print([dato]); - para imprimir los datos.

Se debe mencionar que solo recibe datos en ASCII, la librería hace una conversión interna al tipo de dato usado, abajo se explican un poco más a fondo las funciones de la librería que se usarán.

Serial.begin([velocidad]);

- Se usa para inicializar la comunicación.
- Abre el puerto serie con la velocidad indicada.
- La velocidad está dada en baudios, baudrate.
- Ambos extremos deben usar los mismos baudios.
    - Serial.begin(9600) ;

Serial.available();

- Para poder saber si hay datos en el buffer:
    - Devuelve el número de bytes en el buffer.
    - Envía un cero si no hay datos.
- Podemos usarlo con un if:
    - if(Serial.available() > 0) {//ejecucion}

Serial.read();

- Para leer los datos en el buffer:
    - Devuelve el primer byte disponible en el buffer.
    - Cero o -1 si no hay datos.
- Podemos usarlo en una asignación:
    - dato = Serial.read();

Serial.print(dato); o Serial.println(dato);

- Para enviar un dato:
    - Envía el dato por el serial.
    - El dato puede ser de cualquier tipo.
    - Lo envía como ASCII.

Como siempre se ha mencionado, esta es la base para el aprendizaje, si el lector quiere más información la podrá encontrar en la página oficial de Arduino, en las siguientes prácticas se mostrará cómo usar la comunicación serial, si usted une esto con lo ya aprendido podrá realizar muchos proyectos, por ejemplo control de domótica de los aparatos eléctricos de su casa, aplicando esto y control de cargas.

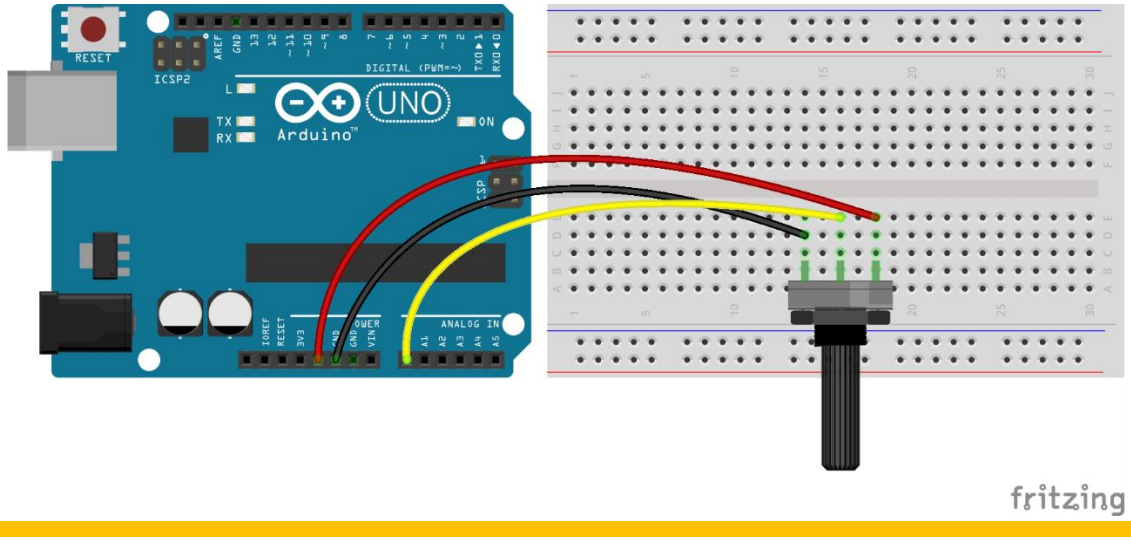
4.6.1 IMPRIMIR DATOS POR EL MONITOR SERIE

En esta práctica leeremos un potenciómetro e imprimiremos los datos, esto ya se vio en la parte se sensores analógicos así que será muy fácil de entender, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 Potenciómetro de 10k
Protoboard
Jumpers

```
#define POT A0
int valor;

void setup() {
  Serial.begin(9600);
  pinMode(POT,INPUT);

}

void loop() {
  valor = analogRead(POT);
  Serial.print("El valor es: ");
  Serial.println(valor);
  delay(50);
}
```

Circuito:



Explicación:

El código ya se vio en los temas anteriores, aquí solo se explicarán las partes importantes. En la parte del **setup** donde se inicializa todo lo necesario, se usa **Serial.begin(9600)**, como bien esto ya se mencionó para inicializar y cargar la comunicación serial a la velocidad de **9600 baudios**. Ya en el **loop** usamos **Serial.print**, esta función la utilizamos para imprimir un mensaje o enviar los datos por medio del **serial**, en este caso al monitor serie, después usamos **Serial.println** para imprimir los valores del potenciómetro, la única diferencia con **print y println** es que **println** también envía un **enter**.

Ahora abriremos el monitor serial, verán que abajo dice **9600 baudio**, recordemos que al principio del tema se mencionó que ambos extremos deben de tener la misma velocidad, al poner **Serial.begin** en el **setup** se configura el **Arduino y su microcontrolador** a la velocidad de **9600**, ahora el **otro extremo** será la **computadora** en este caso se escoge nuevamente **9600 baudios** en el monitor serial.

En esta práctica se enseñó a enviar datos usando la comunicación serial, en la próxima verán cómo leer datos desde el monitor serial.

4.6.2 RECIBIR DATOS POR EL PUERTO SERIE

En esta práctica se ensaña como usar el puerto serie pasa recibir datos, como siempre se menciona esta es la base del aprendizaje, los datos se recibirán a través del monitor serial, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO

```
void setup() {

  Serial.begin(9600);

}

void loop() {

  if(Serial.available() > 0){

    char dato = Serial.read();
    Serial.print("Recibi un: ");
    Serial.println(dato);

    }
}
```

Explicación:

Como pueden ver no hay ningún circuito pues no es necesario, ya que solo usaremos el monitor serial, en el código se inicializa el serial a 9600 baudios como ya se había hecho anteriormente. Ahora lo interesante está en el **loop**, usamos una sentencia de control **if** dentro del if, ponemos la condición **Serial.available() > 0**, esto nos quiere decir que si se envían datos o hay datos en el serial entra en la condición del **if** y se ejecuta la sentencia que está dentro, de lo contrario no hace nada, después creamos una variable del tipo char llamada **dato**, recordemos que el serial solo recibe char y hace el casteo interno, en esta variable llamada **dato** se almacenarán los valores correspondiente al buffer usando **Serial.read()**, después se imprimen con la sintaxis **Serial.print**, empleando el monitor serial podemos introducir los datos y verlos cuando se imprimen.

Como pudieron percatarse, es sumamente fácil leer los datos recibidos por el serial y si esto se une a la sentencia de control, podemos hacer proyectos muy interesantes como en la práctica que verán a continuación.

4.6.3 COMUNICACIÓN SERIAL CON EL BLUETOOTH

En esta práctica se verá la comunicación serial con el módulo bluetooth HC-06, también podrán hacerlo con el HC-05, les recordamos que esta es la base para que aprendan a usarlo, también es necesario que carguen el programa antes de conectar el módulo a los pines RX y TX del Arduino, ya que si no se hace esto, el programa no se cargará, abajo se muestra el código, material y explicación de la práctica.

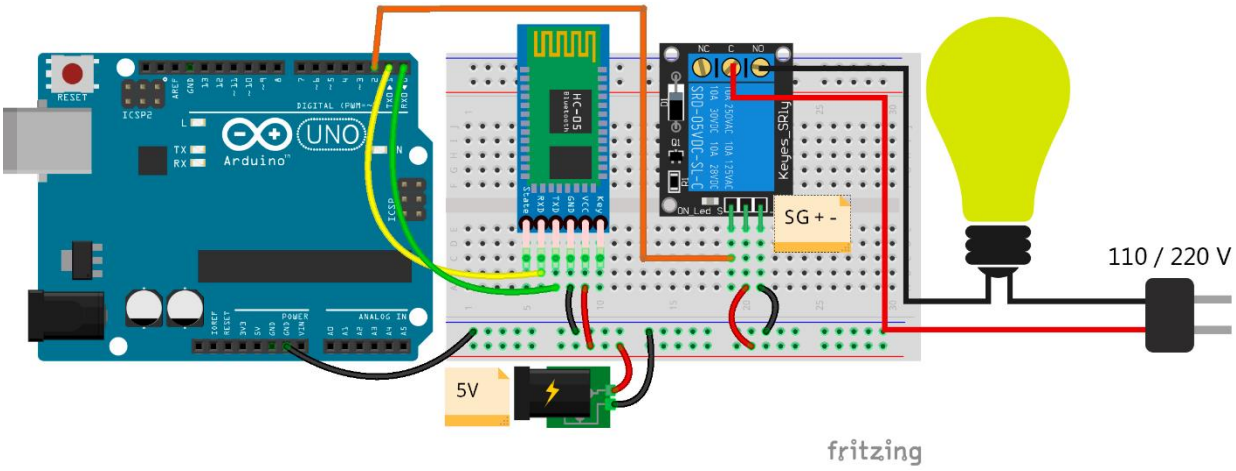
La aplicación que se usará la pueden bajar de este enlace: https://goo.gl/74mzqB

MATERIALES
1 ARDUINO
1 HC-06 o HC-05
Jumpers
Protoboard
1 Módulo Relay
1 Fuente de 5V

```
#define FOCO 2

char dato;

void setup() {
  Serial.begin(9600);
  pinMode(FOCO, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
```

```
    dato = Serial.read();
  }
  switch (dato) {
    case 'A':
      digitalWrite(FOCO, HIGH);
      break;
    case 'B':
      digitalWrite(FOCO, LOW);
      break;
    default:
      digitalWrite(FOCO, LOW);
      break;
  }
}
```

Circuito:



Explicación:

Para esta práctica se quiso poner el ejemplo de prender un foco mediante una señal enviada por bluetooth desde nuestro teléfono móvil, cabe recalcar que el módulo bluetooth debe de estar apareado con su teléfono.

Como pueden observar hay varias cosas interesantes, como el lector ya debió aprender lo esencial no se entrará en detalles. Se realizó un código normal, se definió la inicialización del serial a 9600 baudios pues también es la velocidad de nuestro módulo bluetooth, recuerde que ambos extremos deben de tener la misma velocidad. En el setup está lo interesante, la parte del **Serial.available** ya se vio y se explicó en el tema anterior, ya que con esta condición dentro del **if** detectará si tenemos datos, los datos que se enviarán son la letra **A** y **B**, recuerden que el serial solo acepta ASCII y por eso se usan caracteres.

Ahora, si hay datos entrará dentro del **if** y esos datos se le asignan a la variable **dato**, aquí está lo interesante, usamos un **switch case** que se explicó en los primeros temas, el **switch** recibe como parámetro de casos la variable **dato** y es ahí cuando entra en los **case**, recuerde que usamos char y van

entre comillas simples **‘’**, si se recibe la letra **A** entonces enviamos un pulso alto para activar nuestro relevador y se prende el foco, permanece prendido hasta que se envíe la letra **B**, será cuando se envíe un pulso bajo al relevador.

También se usa un default para que no haya problemas de ruido y de acciones por si solas del relevador, como podrá observar es muy fácil hacer eso y controlar cargas, motores, etc., claro aplicando lo anterior visto. Si usted amplía los casos y usa un módulo con un mayor número de relevadores, podrá controlar muchísimas cosas en su casa o trabajo, obviamente también habrá que diseñar una aplicación móvil, claro esto se le deja al lector ya que también debe de tener la iniciativa de investigar por sí solo.

Con este tema finaliza el aprendizaje de comunicación con Arduino, el siguiente y último tema será para mostrar y visualizar datos tanto en una lcd como un Display, después de este tema se inicia el Capítulo 5 en donde se harán proyectos más avanzados.

Video [tutorial]: *[Curso Arduino Básico - #9] Comunicación con Arduino*

4.7 DESPLEGANDO INFORMACIÓN CON ARDUINO

En esta parte se enseña a usar el LCD 16x2 y el Display de 7 segmentos para mostrar los datos de nuestros sensores, mensajes, etc.

La LCD

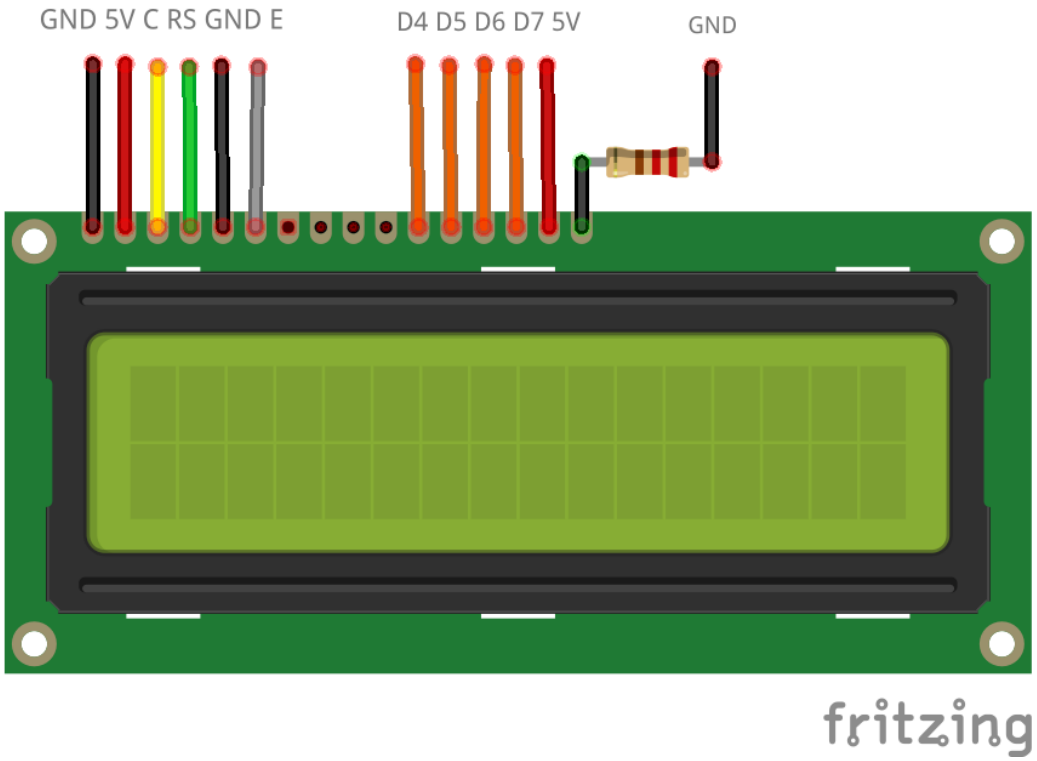
Las pantallas de cristal líquido nos servirán para mostrar datos acerca de nuestros programas, ya sean mensajes o mediciones que nos mandan nuestros sensores u otra información que nosotros requerimos, la configuración que se verá con la LCD es la clásica, se le deja al lector la investigación de la LCD con I2C, enseguida se muestran la funciones necesarias para la LCD y sus pines de conexión, recordemos que para usar la LCD es necesario importar la librería de la LCD que será **#include <LiquidCrystal.h>.**

Funciones:

Función	Descripción
LiquidCrystal [nombre](rs, en, d4, d5, d6, d7);	**Se inicializa la lcd y se le asigna un nombre, además recibe como parámetro los pines donde se ha conectado respectivamente**
[nombre].begin([columnas], [filas]);	**Función donde declaramos de qué tamaño es nuestra lcd**
[nombre].print([mensaje/dato]);	**Función para imprimir un mensaje o dato**
[nombre].clear();	**Función para limpiar la lcd**
[nombre].write([dato]);	**Función para imprimir un dato en la lcd**

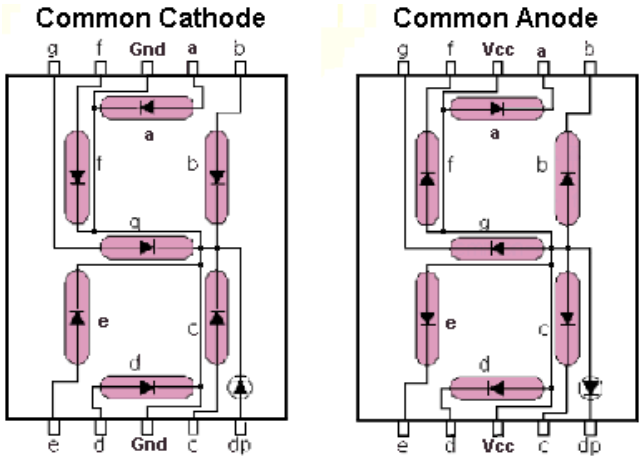
[nombre].setCursor([columna],[fila]);	**Función para posicionar el mensaje o dato que desea mostrar**

Conexión:



El Display de 7 segmentos

Los Display de 7 segmentos son muy usados en la electrónica para mostrar datos numéricos, aunque también sirven para mostrar mensajes. Existen dos tipos de Display de 7 segmentos, ánodo común o cátodo común, abajo se muestran los diagramas de ambos y sus tablas para obtener el número deseado:



Cátodo	a	b	c	d	e	f	g	dp		Ánodo	a	b	c	d	e	f	g	dp
0	1	1	1	1	1	1	0	0		0	0	0	0	0	0	0	1	1
1	0	1	1	0	0	0	0	0		1	1	0	0	1	1	1	1	1
2	1	1	0	1	1	0	1	0		2	0	0	1	0	0	1	0	1
3	1	1	1	1	0	0	1	0		3	0	0	0	0	1	1	0	1
4	0	1	1	0	0	1	1	0		4	1	0	0	1	1	0	0	1
5	1	0	1	1	0	1	1	0		5	0	1	0	0	1	0	0	1
6	1	0	1	1	1	1	1	0		6	0	1	0	0	0	0	0	1
7	1	1	1	0	0	0	0	0		7	0	0	0	1	1	1	1	1
8	1	1	1	1	1	1	1	0		8	0	0	0	0	0	0	0	1
9	1	1	1	0	0	1	1	0		9	0	0	0	1	1	0	0	1

Como puede ver es muy simple entender los diagramas, en cátodo común para que prenda cada uno de los segmentos tiene que enviarse un alto y en ánodo común se envía un bajo.

La tabla indica que, si queremos que se muestre en nuestro Display ánodo común, entonces el número 1 se tiene que enviar a los segmentos 1001110, como ya se vio anteriormente el **1** representa un **HIGH** y el **0** un **LOW**.

Para mostrar números largos de más de un digito se usan Display de 2, 3, 4, etc. dígitos, pero se usa el multiplexado que en simples palabras engaña a la vista prendiendo y apagando el Display muy rápidamente para que se vea que se forma un número compuesto, aunque en verdad primero saca un numero en todos los Display pero se apaga en los que no debe aparecer ese número y así sucesivamente con los demás.

Para las prácticas que se mostrarán es necesario el uso de un módulo Display de 7 segmentos y 4 dígitos, para que pueda apreciar el uso correctamente, este módulo usted puede hacerlo de manera casera siguiendo el siguiente tutorial:

Video **[tutorial]:** *[Tutorial Electronica] Circuitos Utiles - Modulo Display 7 Segmentos*

Dependiendo del tipo de Display que use será el tipo de manejo, en este caso se usará un el módulo que es ánodo común.

4.7.1 USO DE LA LCD DE 16 X 2

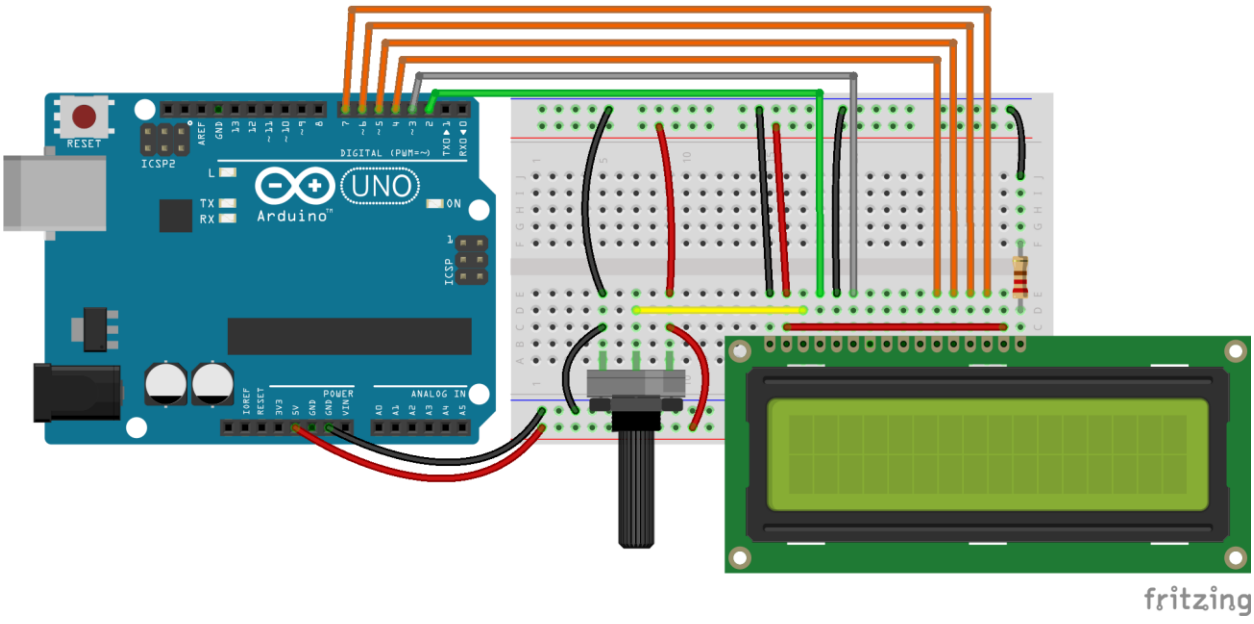
En esta práctica se verá con un ejemplo simple el uso de la LCD 16 x 2, pues solo mostraremos dos mensajes en al LCD, abajo se muestra el código, material y explicación de la práctica.

Recuerde que esto es la base del aprendizaje, queda en manos del lector investigar a fondo el uso de ésta para poder obtener el 100% de su uso.

MATERIALES
1 ARDUINO
1 LCD 16x2
1 Potenciómetro de 10k
1 Resistencia de 220
Jumpers
Protoboard

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(4, 0);
  lcd.print("Hola Mundo");
  lcd.setCursor(3, 1);
  lcd.print("Arduino UNO!");
}
```

Circuito:



Explicación:

Como pueden observar las conexiones son muy simples, usamos los pines del 2 al 7 para la conexión de nuestra LCD y utilizamos un potenciómetro para controlar el contraste de nuestra LCD.

En nuestro código cargamos al librería de la LCD usando **#include <LiquidCrystal.h>**, después definimos la LCD nombrándola **lcd**, además escribimos en qué pines está conectado cada pin de la LCD, esto se vio en la tabla que está más arriba. En el **setup** iniciamos la LCD con las columnas y filas que tiene, en este caso primero van las columnas y después las filas usando **lcd.begin(16,2),** recuerde que **lcd** es el nombre que se le dio a la LCD.

En el **loop** escribimos nuestro programa de ejecución usando la función **setCursor** con la **lcd**, asignamos que nuestro primer mensaje **"Hola Mundo"** aparezca en la primera fila en la columna 4, pero dirán – ahí hay un cero, así está asignado en la librería la primera fila empieza en **0**, la segunda con **1** y así sucesivamente, después usando **lcd.print** escribimos el mensaje que queremos enviar, debe de ir entre **comillas dobles** o bien pasamos la **variable** con los datos que queremos imprimir, después volvemos a usar **lcd.setCursor** para seleccionar en qué posición se imprimirá un segundo mensaje con **"Arduino UNO!"**.

En esta práctica ya se mencionó lo más esencial para el uso de la LCD y mostrar mensajes de texto, se le deja al lector jugar con más mensajes, espacios y posiciones para mostrar esos mensajes, en la siguiente práctica se verá como mostrar datos de nuestros sensores.

4.7.2 MOSTRAR DATOS EN LA LCD

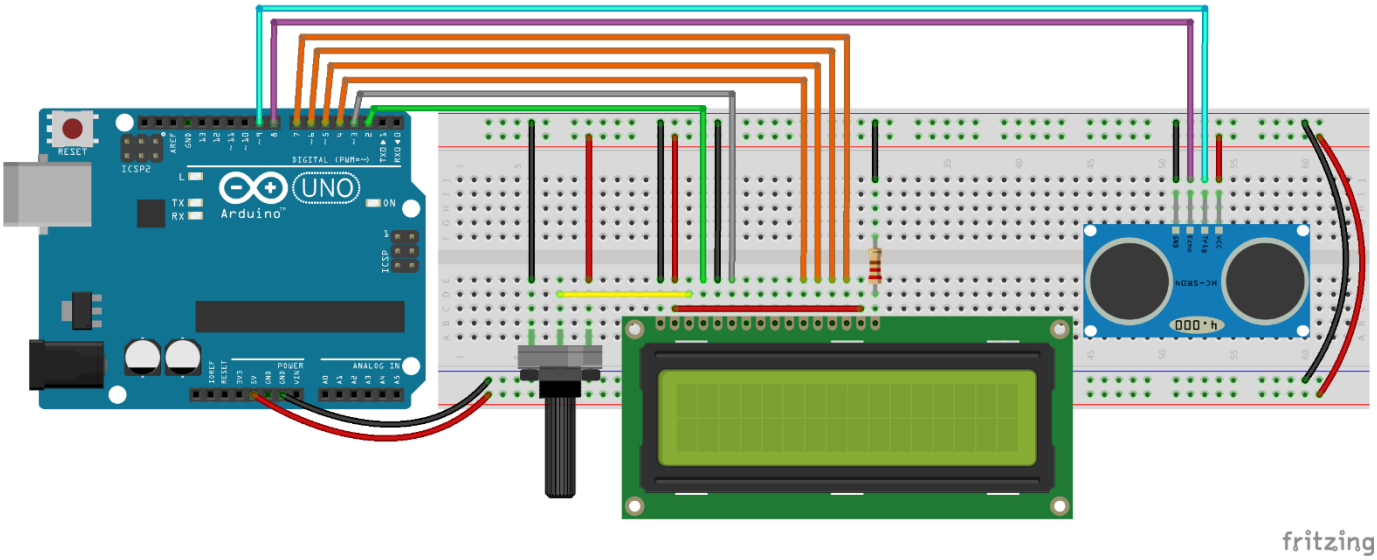
Como lo dice el título, en esta práctica mostraremos lecturas en nuestra LCD de dos de nuestros sensores, en este ejemplo usaremos el sensor digital HC-SR04 que ya se vio en el tema de sensores digitales, así que no se entrará en detalles para la explicación sobre este sensor. Nuestra LCD nos sirve perfectamente para mostrar distintos datos, temperatura, tiempo, humedad, etc., dependiendo de qué queremos que se muestre, como bien ya se ha mencionado bastante, esta práctica solo es la base del aprendizaje, el uso se lo da el lector, abajo se muestra el código, material y explicación de la práctica.

MATERIALES
1 ARDUINO
1 LCD 16x2
1 Potenciómetro de 10k
1 Resistencia de 220
1 HC-SR04
Jumpers
Protoboard

```
#include <LiquidCrystal.h>
#define TRIG 9
```

```
#define ECHO 8

LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

long Tiempo;
float Distancia;

void setup() {
  lcd.begin(16, 2);
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
}

void loop() {
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  Tiempo = (pulseIn(ECHO, HIGH)/2);
  Distancia = float(Tiempo * 0.0343);

  lcd.setCursor(0,0);
  lcd.print("Distancia: ");
  lcd.print(Distancia);
  delay(10);
}
```

Circuito:



Explicación:

El código es bastante simple, primero que nada, incluimos la Liberia de la LCD y se define esta misma, después se definen los pines a utilizar para el sensor ultrasónico y las variables necesarias. En el setup se inicializa la LCD con **lcd.begin** y también los pines, ya en nuestro **loop** se hacen los cálculos para medir la distancia. El uso de la LCD está un poco más abajo, usando **setCursor** queremos que nuestro mensaje aparezca en la primera fila y primera columna, después usamos **lcd.print** e imprimimos **"Distancia"**, volvemos a utilizar **lcd.print** pero ahora pasamos como parámetro la variable **Distancia** a donde se almacena la distancia que nos arroja el sensor ultrasónico, como ven, ahora para imprimir la distancia no usamos **setCursor**, si no se usa la impresión será seguida cuando se definió el primer **setCursor**, después simplemente se usa un delay de 10 para tener un tiempo en que cambien las medidas de la distancia.

Con esta práctica concluimos la parte del uso de la LCD, ahora en la siguiente práctica se mostrará el uso del Display de 7 segmentos.

4.7.3 MOSTRAR DATOS DISPLAY DE 7 SEGMENTOS

En esta y última práctica del libro se enseñará a mostrar datos numéricos en un Display de 7 segmentos de 4 dígitos ánodo común, usando el multiplexeo, se le deja al lector investigar más sobre este tema. Esta será la última práctica del libro, por consiguiente sigue el Capítulo 5 que contendrá 3 prácticas avanzadas, las cuales el lector debe elaborar por sí solo, aunque tendrá un poco de ayuda. Abajo se muestra el código, material y explicación de la práctica.

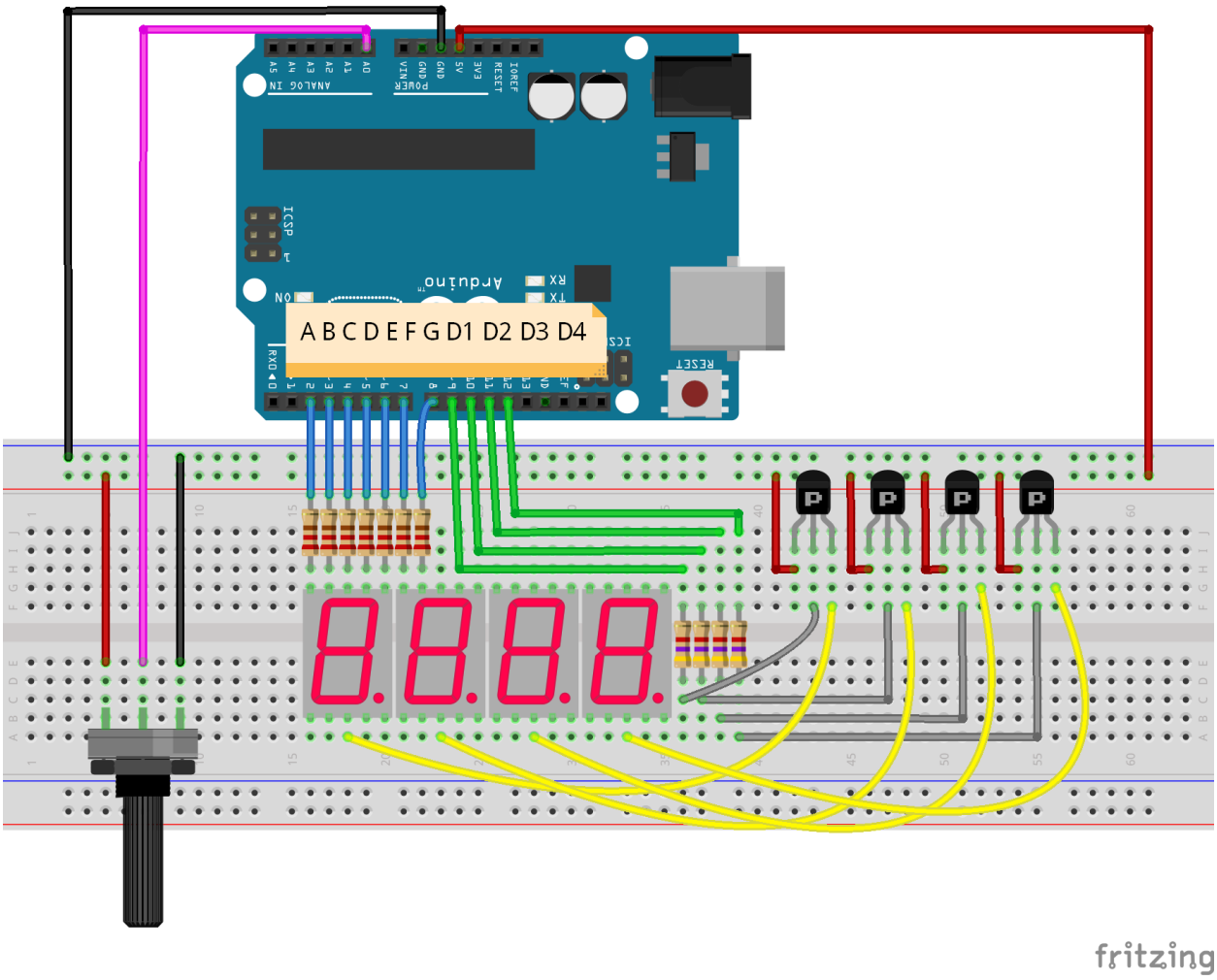
MATERIALES
1 ARDUINO
1 Módulo Display de 7 segmentos 4 dígitos ánodo común
1 Potenciómetro de 10k
Jumpers
Protoboard

```
#define POT A0

#define D1 9
#define D2 10
#define D3 11
#define D4 12

const int segs[7] = {
  2, //A
  3, //B
  4, //C
  5, //D
  6, //E
```

```
  7, //F
  8 //G
};

const byte numbers[] = {
  0b1000000, //0
  0b1111001, //1
  0b0100100, //2
  0b0110000, //3
  0b0011001, //4
  0b0010010, //5
  0b0000010, //6
  0b1111000, //7
  0b0000000, //8
  0b0010000 //9
};

void setup() {
  pinMode(POT, INPUT);
  pinMode(D1, OUTPUT);
  pinMode(D2, OUTPUT);
  pinMode(D3, OUTPUT);
  pinMode(D4, OUTPUT);
  for (int i = 2; i < 9; i++) {
    pinMode(i, OUTPUT);
  }
}

void loop() {
  int Val = analogRead(POT);

  MostrarNum(Val);
}

void MostrarNum(int num) {
  int digit1 = (num / 1000) % 10;
  int digit2 = (num / 100) % 10;
  int digit3 = (num / 10) % 10;
  int digit4 = num % 10;

  Digito1(numbers[digit1]);
  delay(2);
  Digito2(numbers[digit2]);
  delay(2);
  Digito3(numbers[digit3]);
  delay(2);
  Digito4(numbers[digit4]);
```

```
  delay(2);
}

void Digito1(byte n) {
  digitalWrite(D1, LOW);
  digitalWrite(D2, HIGH);
  digitalWrite(D3, HIGH);
  digitalWrite(D4, HIGH);
  segmentos(n);
}

void Digito2(byte n) {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, LOW);
  digitalWrite(D3, HIGH);
  digitalWrite(D4, HIGH);
  segmentos(n);
}

void Digito3(byte n) {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, HIGH);
  digitalWrite(D3, LOW);
  digitalWrite(D4, HIGH);
  segmentos(n);
}

void Digito4(byte n) {
  digitalWrite(D1, HIGH);
  digitalWrite(D2, HIGH);
  digitalWrite(D3, HIGH);
  digitalWrite(D4, LOW);
  segmentos(n);
}

void segmentos(byte n) {
  for (int i = 0; i < 7; i++) {
    int bit = bitRead(n, i);
    digitalWrite(segs[i], bit);
  }
}
```

Circuito:



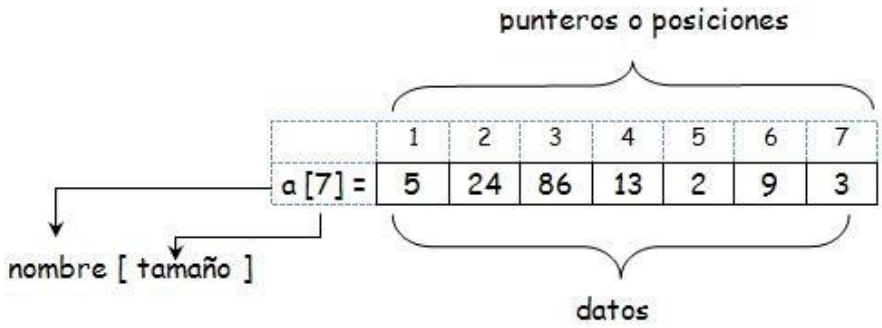
Explicación:

En esta ocasión tenemos un código bastante extenso y algunas partes son complejas, primero que nada se usa el Módulo Display de 7 segmentos 4 dígitos que ya se había recomendado anteriormente.

Primero definimos el pin del potenciómetro que usaremos ya que visualizaremos ese valor en el Display de 7 segmentos, después se definen los pines de cada Display, o sea para que se activen y desactiven (multiplexeo).

Se crea un arreglo del tipo entero llamado **segs** de tamaño 7 con los datos 2, 3, 4, 5, 6, 7, 8 ya que esos serán los pines de Arduino donde se conectarán las letras de nuestro Display, la letra A al 2, la B al 3, etc.

Después creamos de nuevo otro arreglo, esta vez del tipo byte llamado **numbers** ya que guardaremos en byte la formación de números del Display de 7 segmentos, como se vio al principio del capítulo, esta vez se ordena al revés de G hasta A.
Como se mencionó, no entraremos en mucho detalle para la explicación del uso de arreglos, esto se le deja investigar al lector, pero aquí agregamos una imagen para que se entienda un poco mejor:



- Un vector se **define**: nombre [tamaño] arreglo de tipo

Ejemplo: a [7] arreglo de enteros

- Para hacer **referencia**: nombre [posición]

Ejemplo: a [3] = 86
a [5] = 2

Ya en el setup inicializamos los pines tanto de entrada que será el del potenciómetro como de salida que serán los Display y también sus letras, estos los ponemos como salidas usando un **for** como hicimos en las primeras prácticas.

Ahora nos saltamos a la función **MostrarNum**, esta función recibe como parámetro cualquier número entero para que lo mostremos en el Display de 7 segmentos, aquí usamos operaciones matemáticas para descomponer el número y obtener las unidades que se le asigna a la variable **digit4**, obtenemos también las decenas, centenas y las unidades de millar, esta última se le asigna a la variable **digit1**, todas estas deben de ser del tipo entero.

Después estos valores se les asignan al índice del arreglo **numbers** y esto a su vez se le asigna a las funciones **Digito1, Digito2** y así sucesivamente, también usamos un **delay** de dos milisegundos, esto se llama multiplexeo, ya se mencionó que, lo que esto hace es pasarle el mismo número a todos los Display pero se apagan los Display donde no se necesita ver ese número, aunque se hace muy rápidamente que no se puede ver cuando se apagan, si usted quiere ver el proceso de multiplexeo cambie el tiempo del **delay** de dos milisegundos a mil milisegundos.

Ahora, para que funcione eso tenemos las funciones **Digito1, Digito2**, etc., todas similares, que reciben un parámetro del tipo **byte**, pero este parámetro se usa en otra función que se explica más adelante.

Como ya se explicó, dentro de estas funciones enviamos un LOW al Display que queremos que se prenda ya que usamos transistores PNP y a los demás les enviamos un HIGH para que estén apagados.

Dentro de estas funciones tenemos la función **segmentos**, esta función servirá para que se prendan los segmentos del Display, o sea las letras, esta función recibe un parámetro del tipo **byte**, dentro de esta función tenemos un **for** de 0 a 6 que son el número de segmentos de nuestro Display, después en una variable del tipo int llamada **bit** se le asigna un valor usando la función **bitRead**, en simples palabras, esta función entrega el valor dado un byte y el índice que se le pasa como segundo parámetro, por ejemplo:

Tenemos el byte 10101001 que se encuentra en la variable **x** y usando la función bitRead(x, 5) se lo asignamos a la variable **y**, entonces en **y** vamos a tener el valor de 1, ya que se lee de derecha a izquierda.

Después usamos **digitalWrite** para prender los segmentos de cada Display, pasamos como parámetro el arreglo **segs** y bit, que hará que prendan los segmentos de cada Display conectados de los pines 2 a 8.

Como pudieron ver, no es difícil controlar el Display de 7 segmentos, solo es cuestión de pensar cómo se hará el algoritmo para el uso, esto se aprende leyendo múltiples libros de programación y claro, practicando.

Este fue el último tema esencial del libro, a este nivel usted ya debió haber aprendido el uso completo de Arduino, solo queda aplicar todos esos conocimientos y realizar sus proyectos, además se les recomienda seguir estudiando por su cuenta, en la red hay miles de páginas web en donde encontrar información acerca del uso de Arduino y otros microcontroladores, se les recomienda no hacer mal uso del copy/paste, pues al hacer eso usted pierde la oportunidad de aprender, crecer y desarrollar por sí solo, solo queda agradecerles por haber leído y concluido este libro con éxito.

A continuación se muestra el Capítulo 5, donde se crearán 3 prácticas avanzadas para las cuales el código no estará terminado, sino que el lector deberá de concluirlo ya que tendrá los conocimientos necesarios pues ya los ha adquirido leyendo los capítulos anteriores.

Video [tutorial]: *[Curso Arduino Básico - #10] Desplegando informacion*

CAPÍTULO 5

En este capítulo se harán dos prácticas, donde el lector podrá aplicar lo ya aprendido, la primera práctica será el desarrollo de un brazo robótico y su control usando nuestro teléfono móvil, aquí aplicaremos lo aprendido en el uso de motores y en comunicación con Arduino, la segunda práctica será el desarrollo de un temporizador.

5.1 BRAZO ROBÓTICO

En esta práctica se enseñará a controlar un brazo robótico hecho con servomotores y una aplicación móvil, en seguida se enlista el material y los enlaces para descargar los materiales necesarios:

Material:

App: https://goo.gl/RvxMRN

Chasis: https://goo.gl/7PQWsN

Nota: El chasis se debe cortar con láser ya sea en MDF o acrílico de 3mm.

Fuente de 5V y 12V de 2ª (puede ser una fuente atx de pc).

1 Arduino NANO o UNO.

1 Módulo bluetooth HC-05 o HC-06.

4 Servomotores SG90 o MG90.

Jumpers MM, MH.

10 Tuercas M3.

9 Tornillos de 6mm M3.

12 Tornillos de 8mm M3.

3 Tornillos de 10mm M3.

7 Tornillos de 12mm M3.

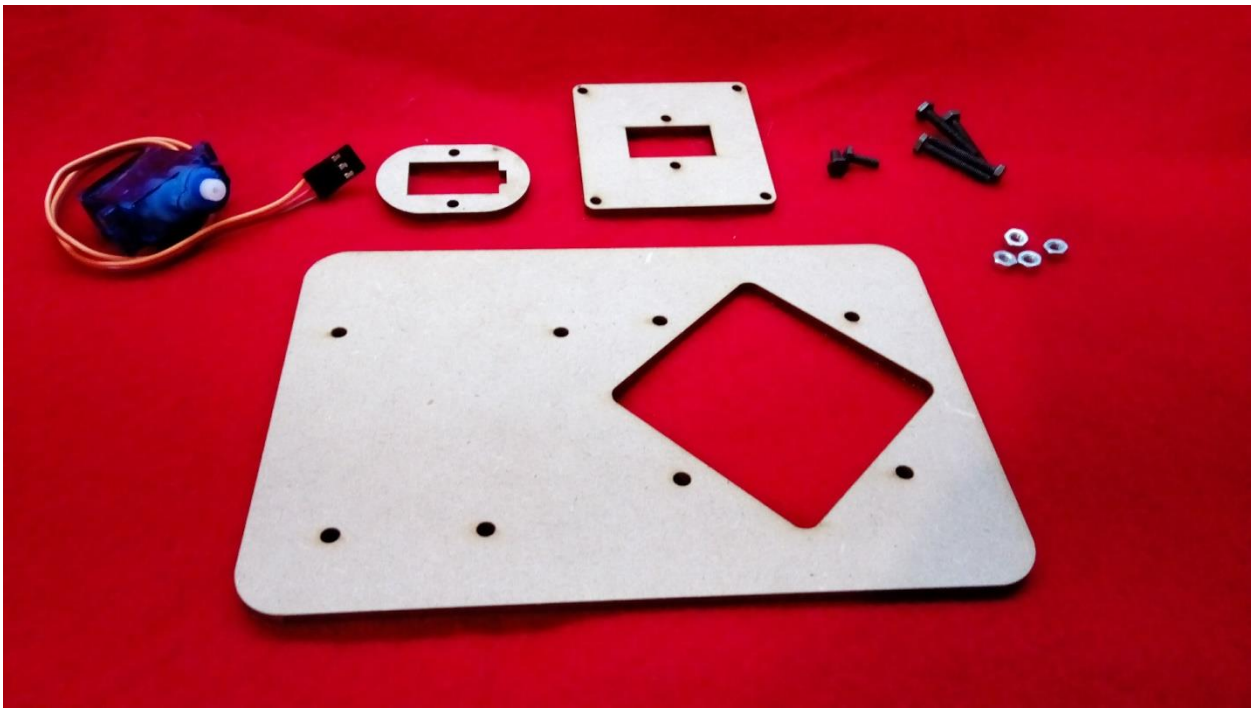
4 Tonillos de 20mm M3.

A continuación se dan las intrusiones para el armado del brazo, antes de comenzar con la programación.

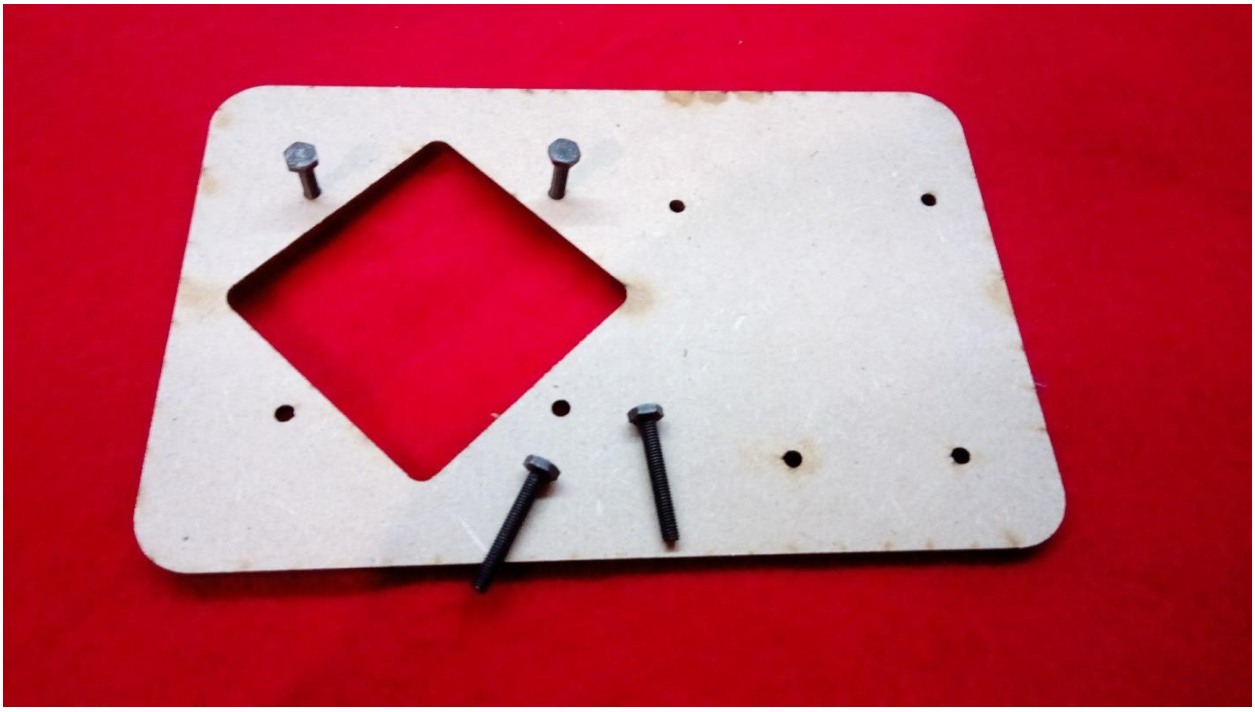
5.1.1 ARMADO DEL BRAZO - BASE

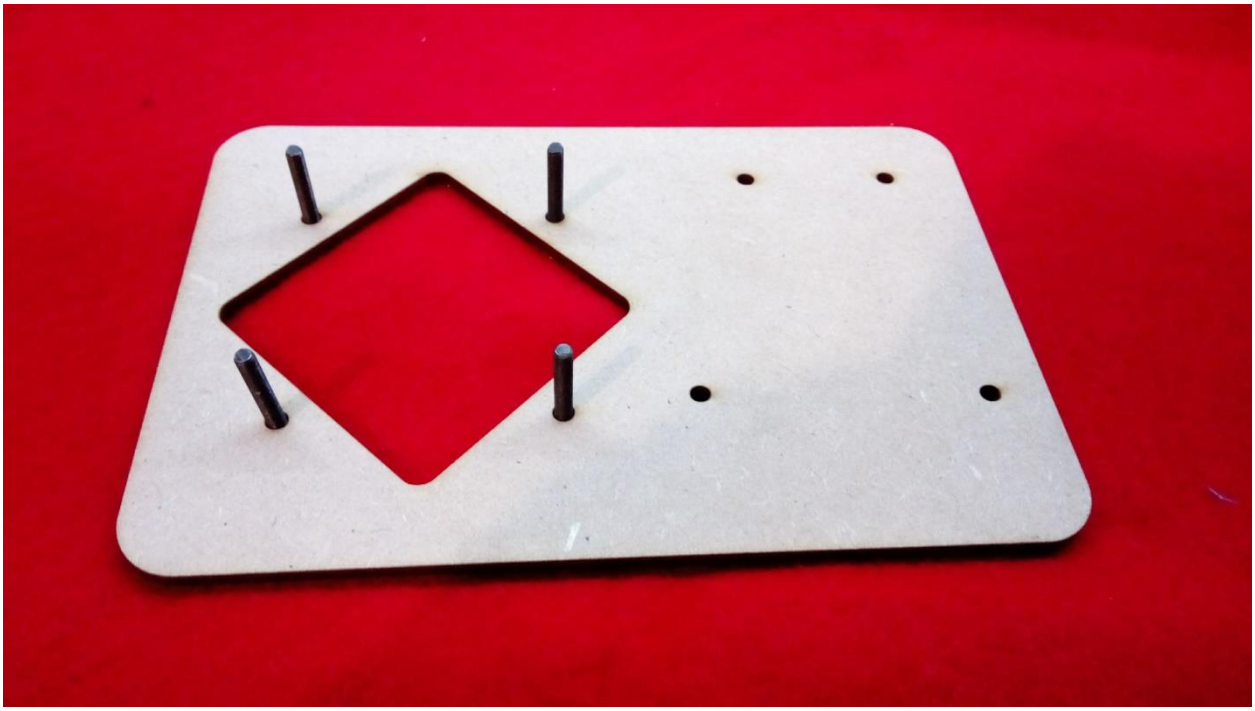
En esta parte se va a mostrar cómo armar la base del brazo, a continuación se lista el material necesario para la misma.

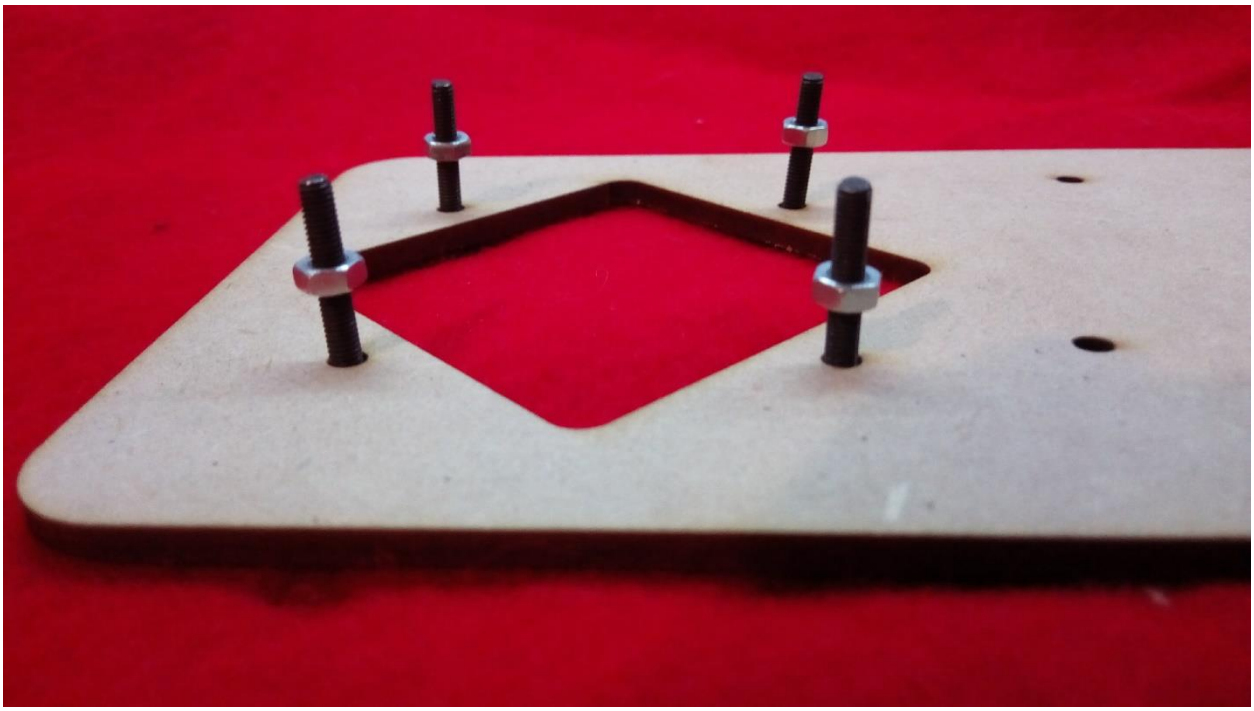
MATERIALES
Base (se muestra en la imagen)
1 Servomotor
4 Tornillos de 20mm
2 Tornillos de 8mm
4 Tuercas
Collar (se muestra en la imagen)



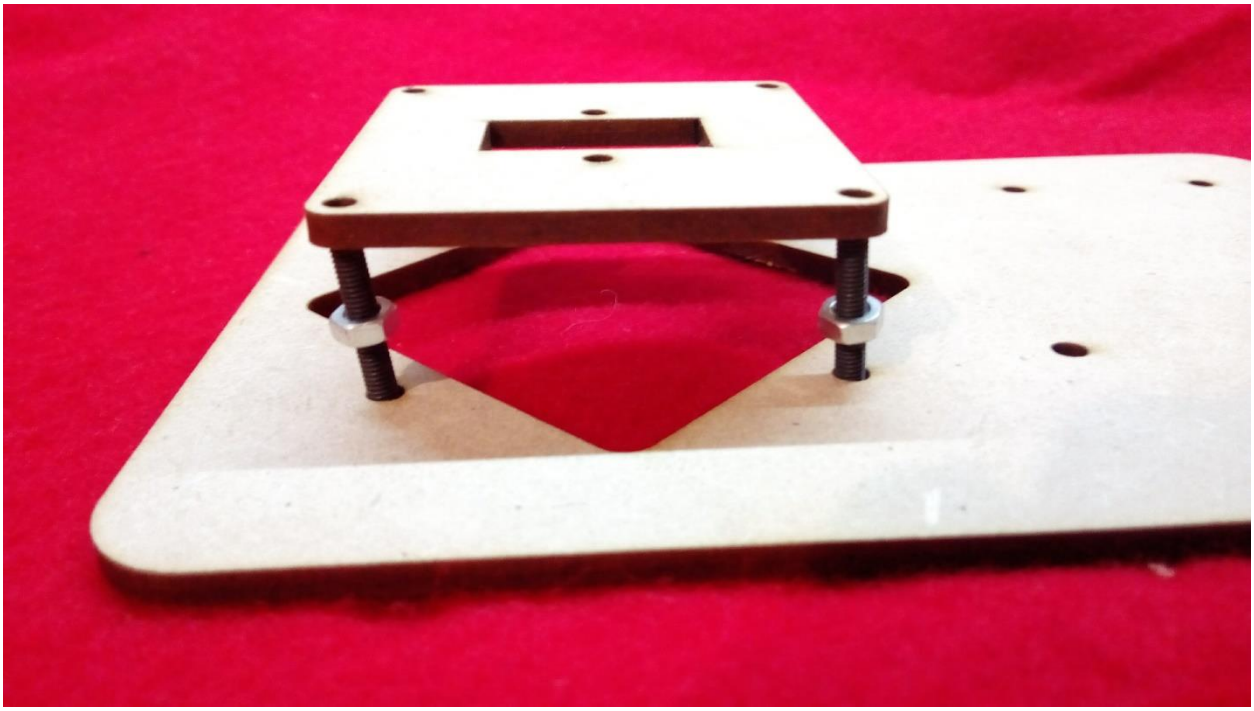
Empiece a insertar los tornillos de 20mm en los orificios alrededor del agujero con forma de cuadrado. Ahora inserte las tuercas en los tornillos de 20mm y gírelas hasta la mitad de arriba hacia abajo.

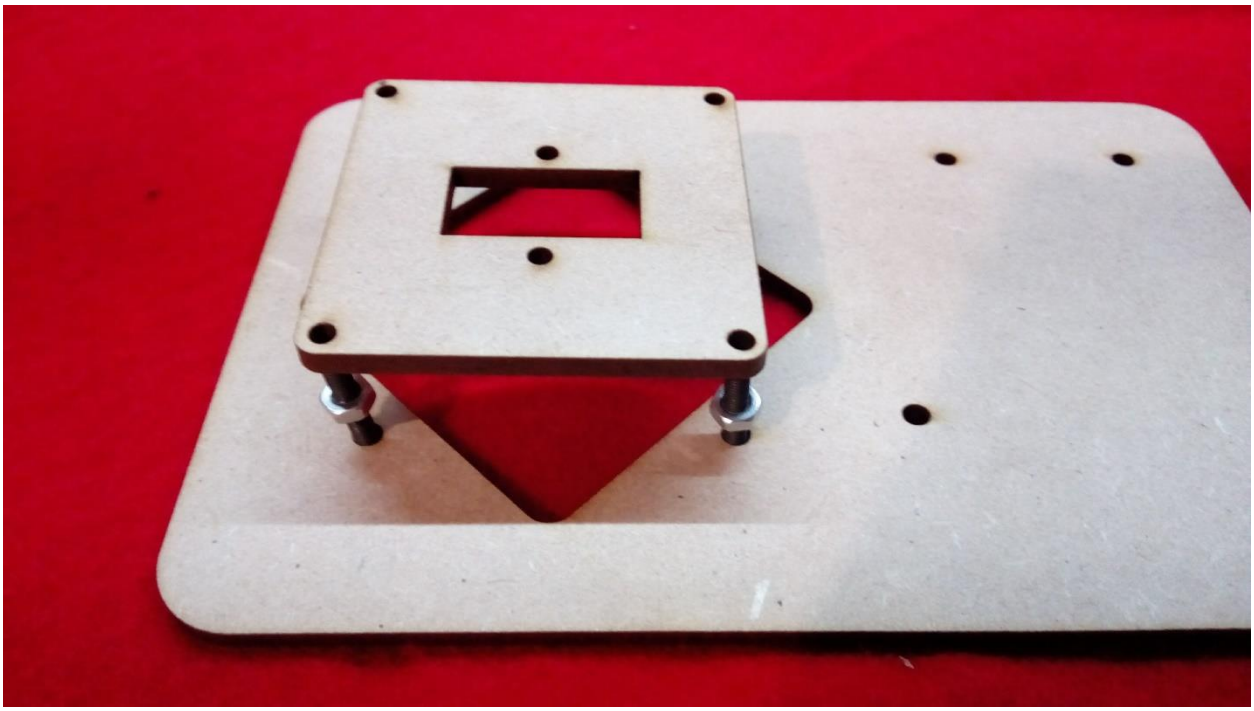


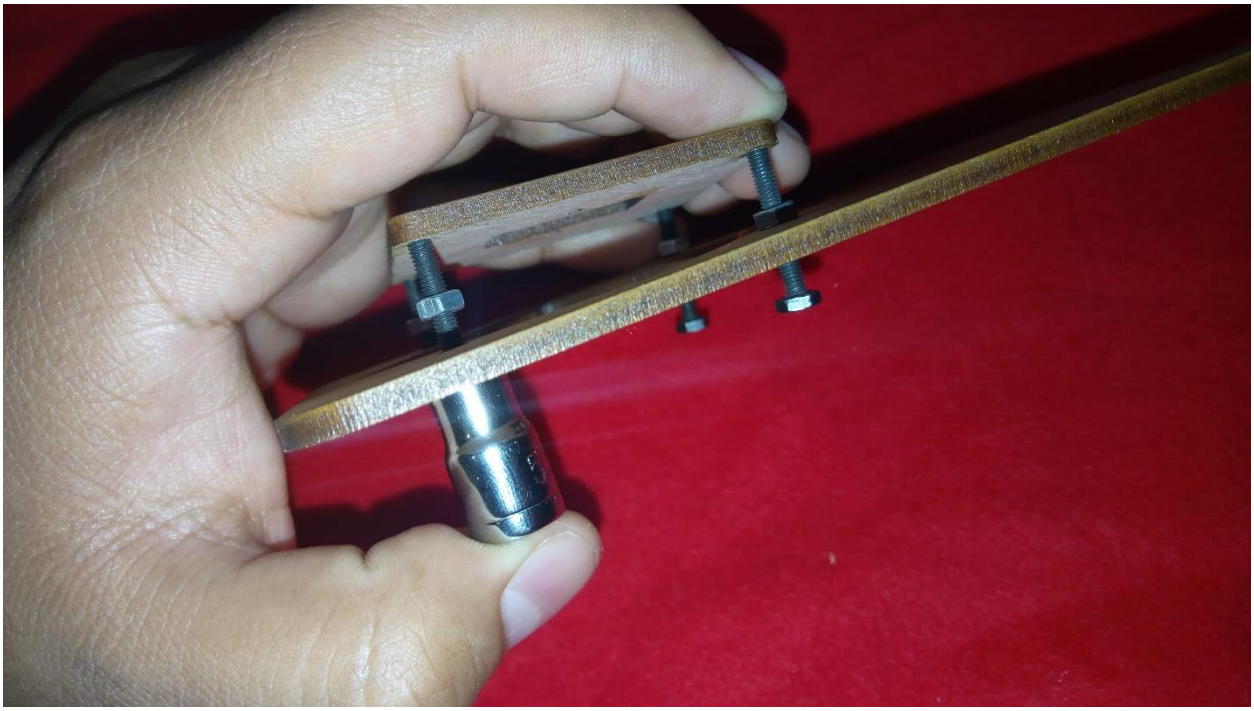


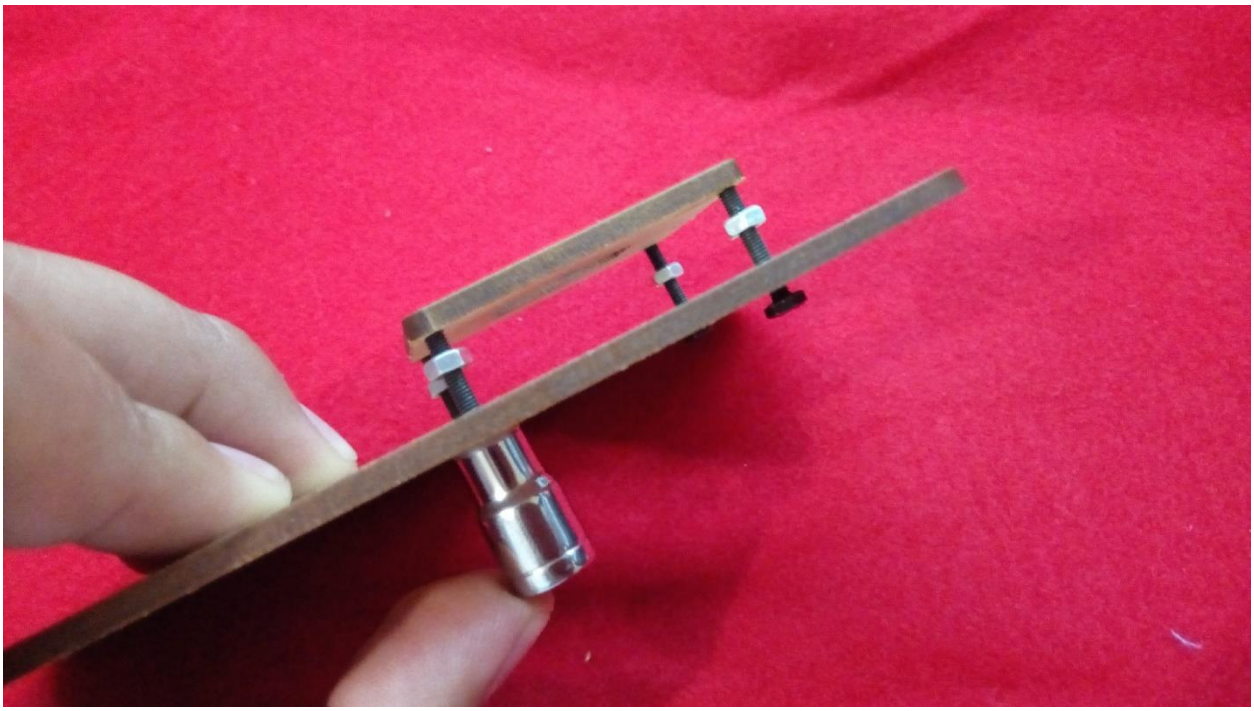


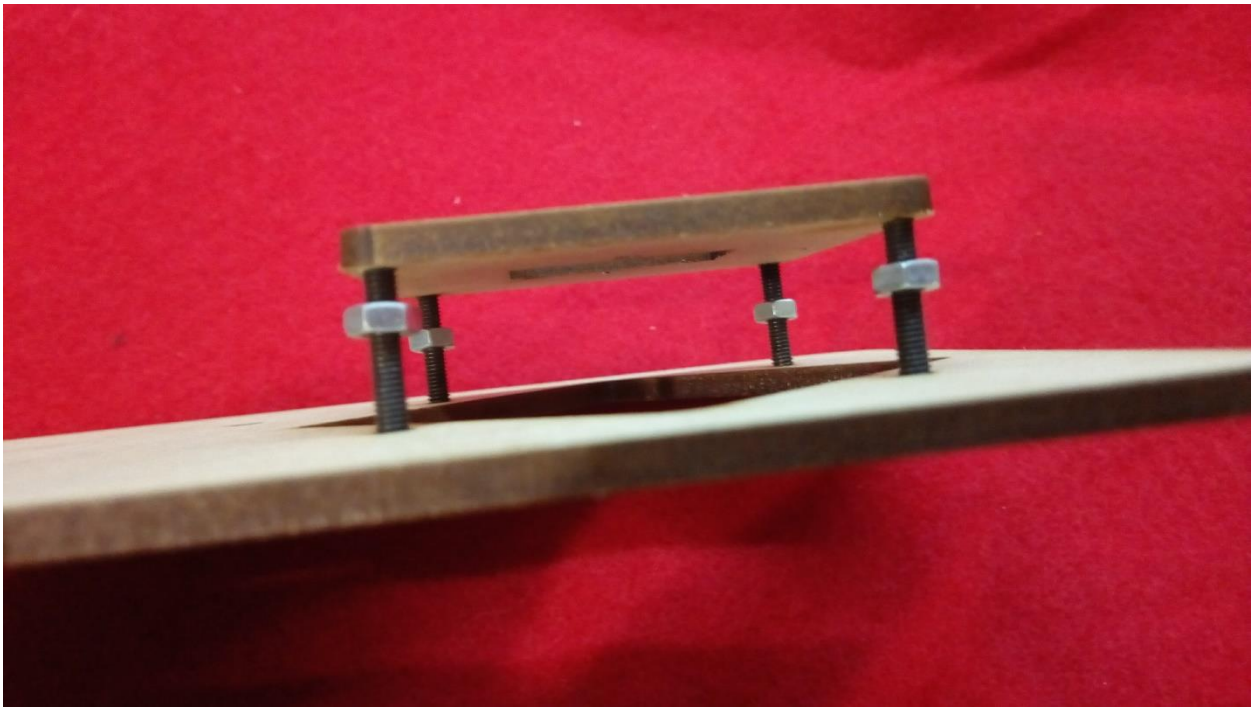
Para añadir el servomotor a la base, tome la pieza cuadrada y colóquela en la parte superior de los tornillos de 20mm con el rectángulo orientado de la misma manera que la base y apriete los tornillos, estos deben de quedar al ras de los agujeros de la parte cuadrada, una vez hecho eso debe de apretar las tuercas debajo de la base (como se ve en las siguientes imágenes).

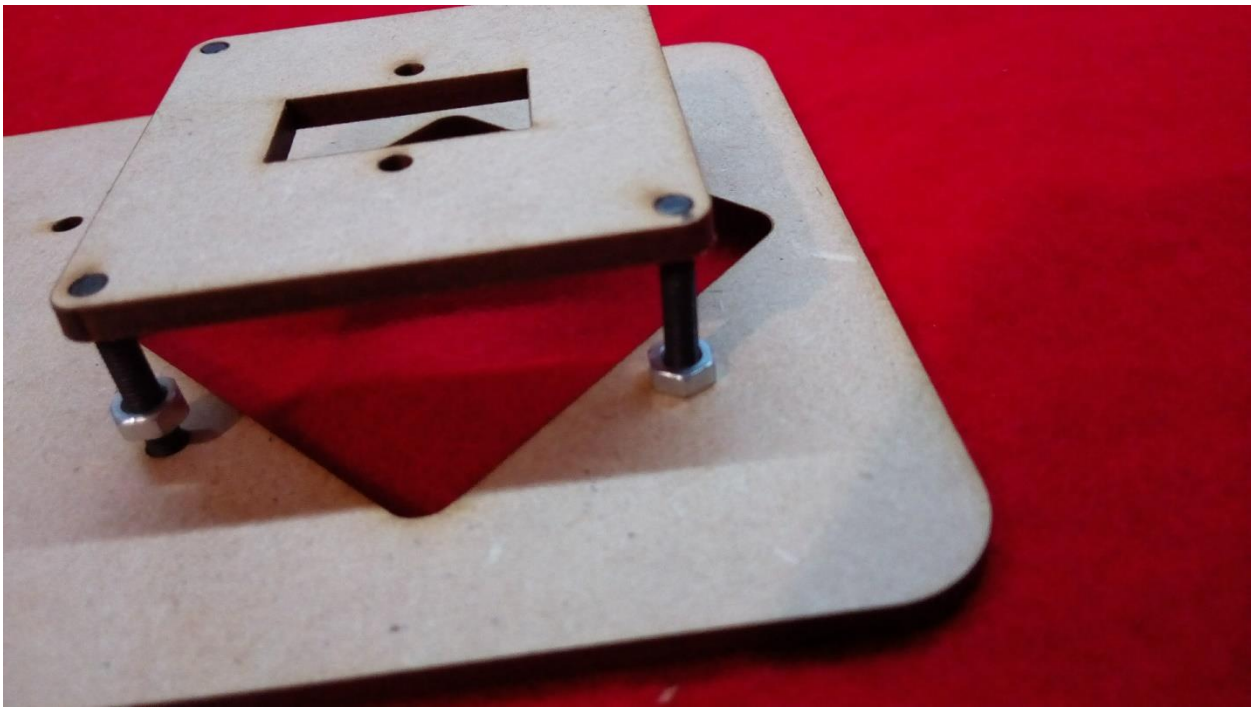


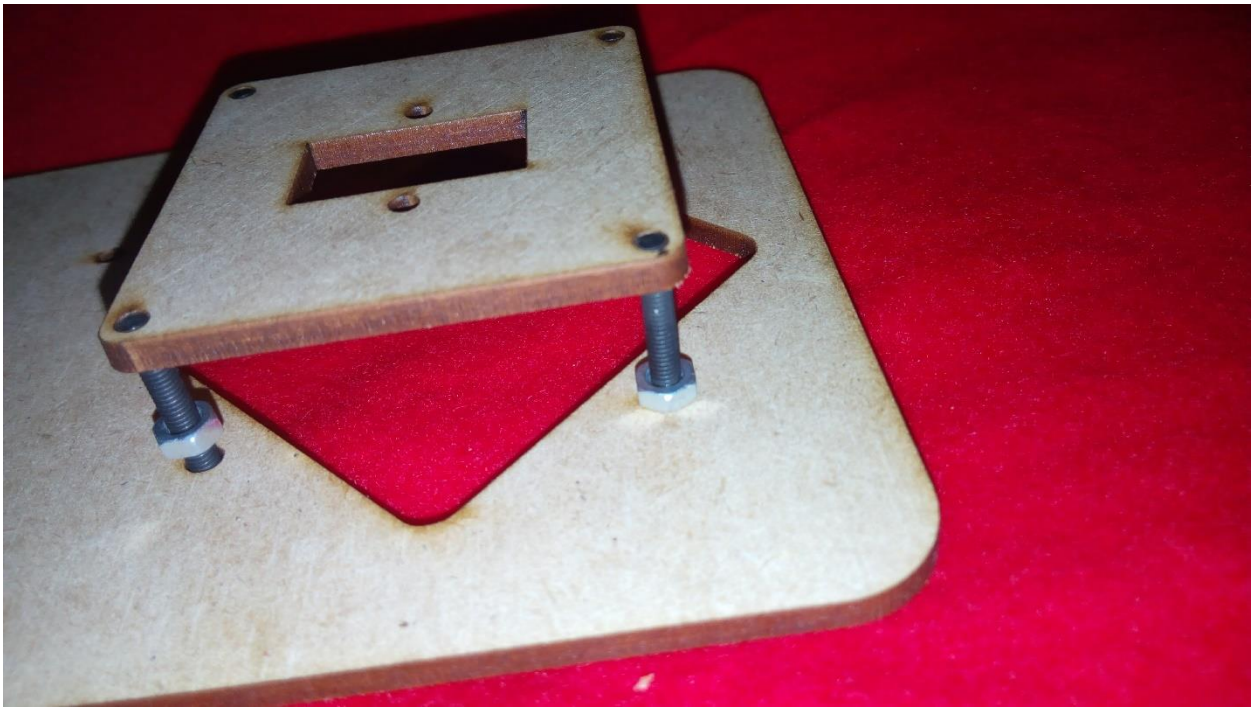


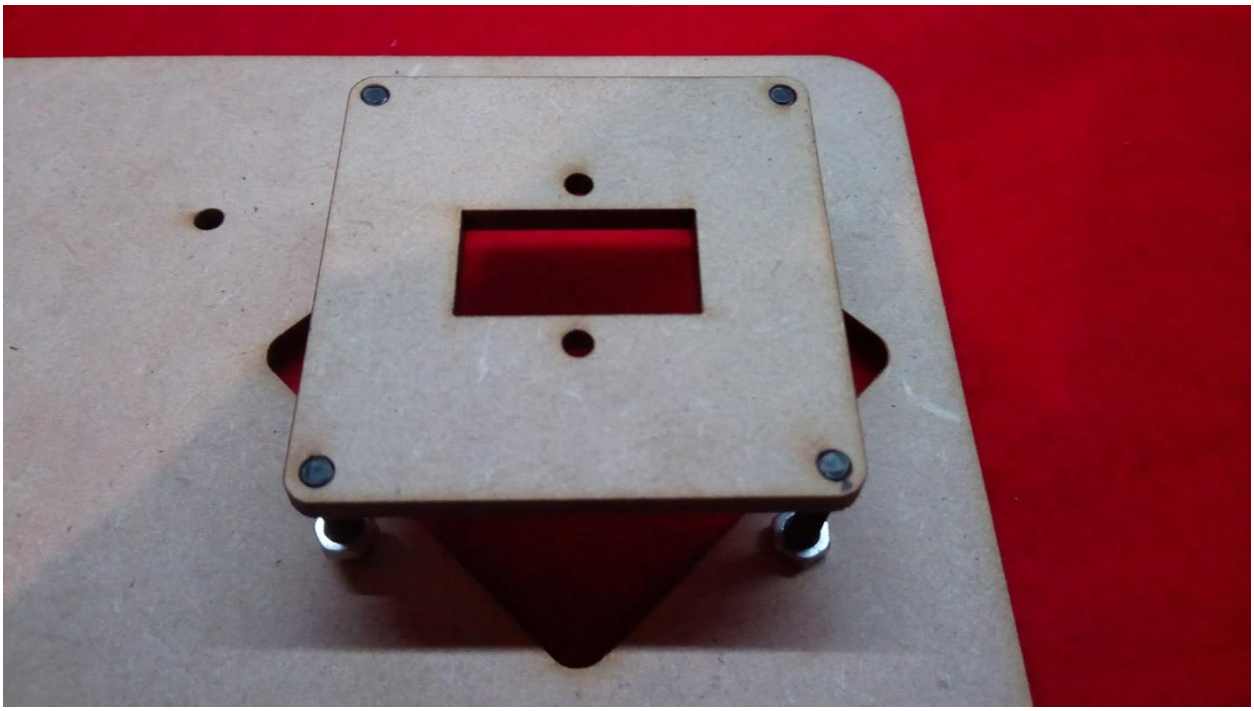






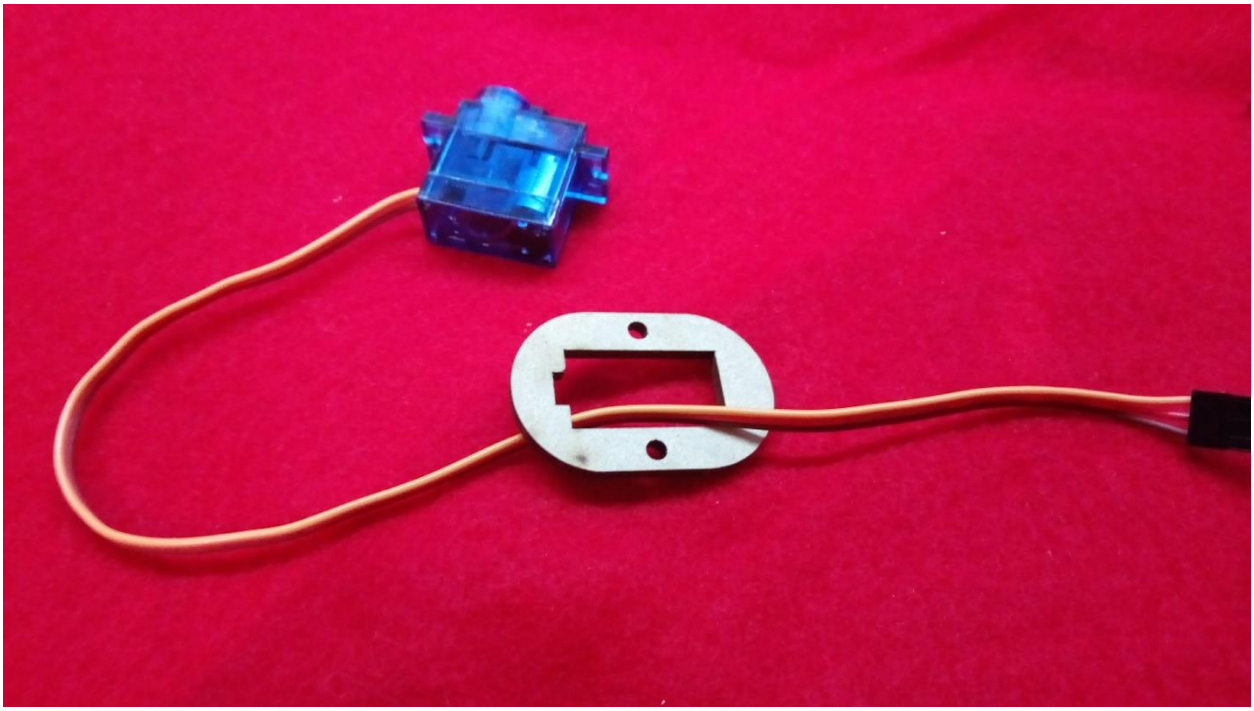


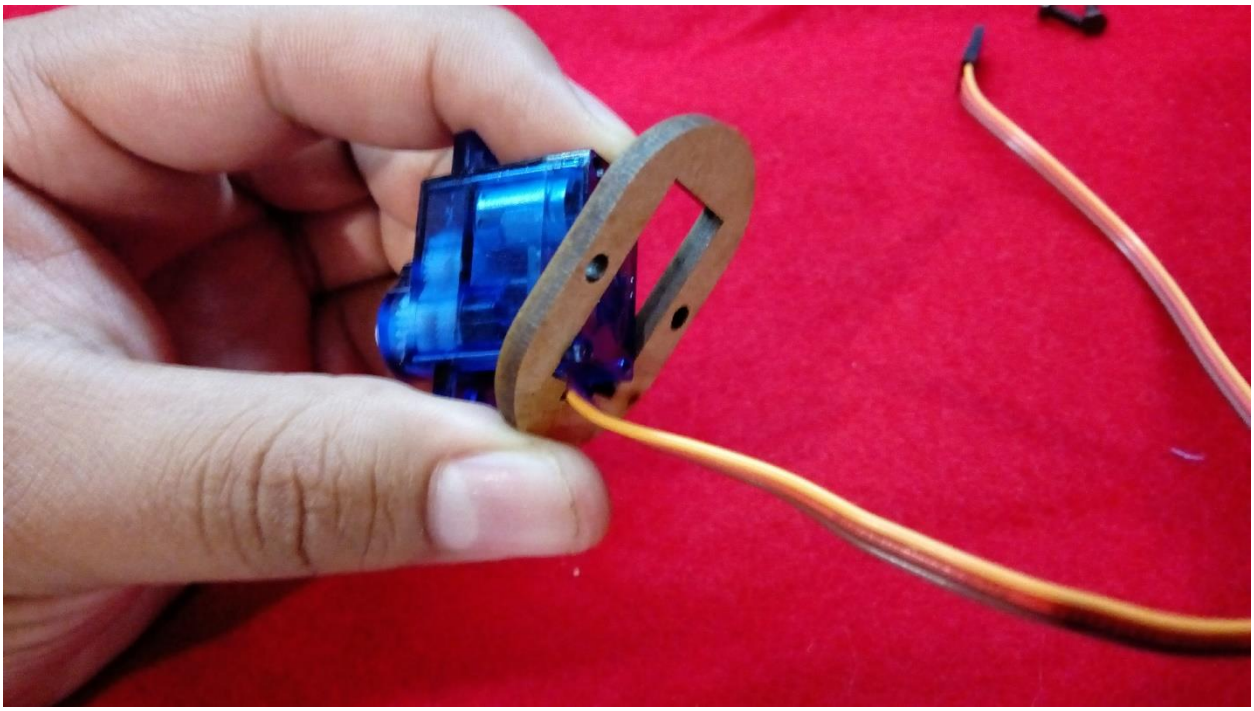


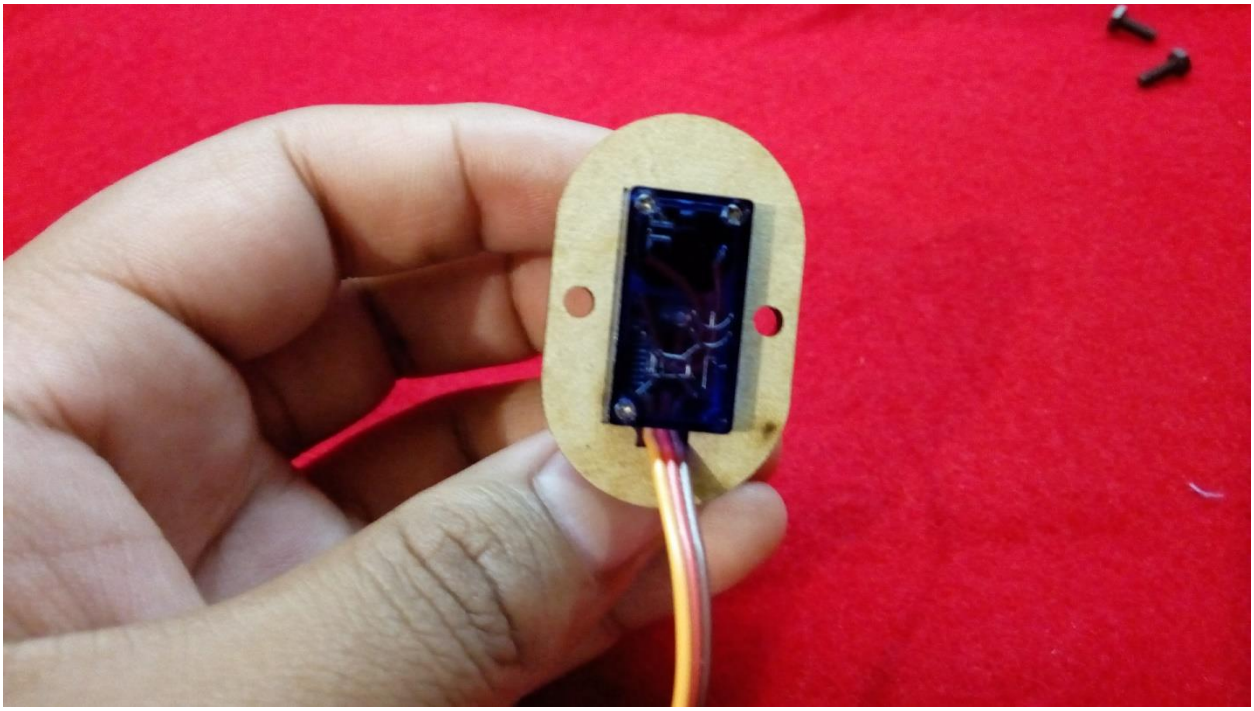


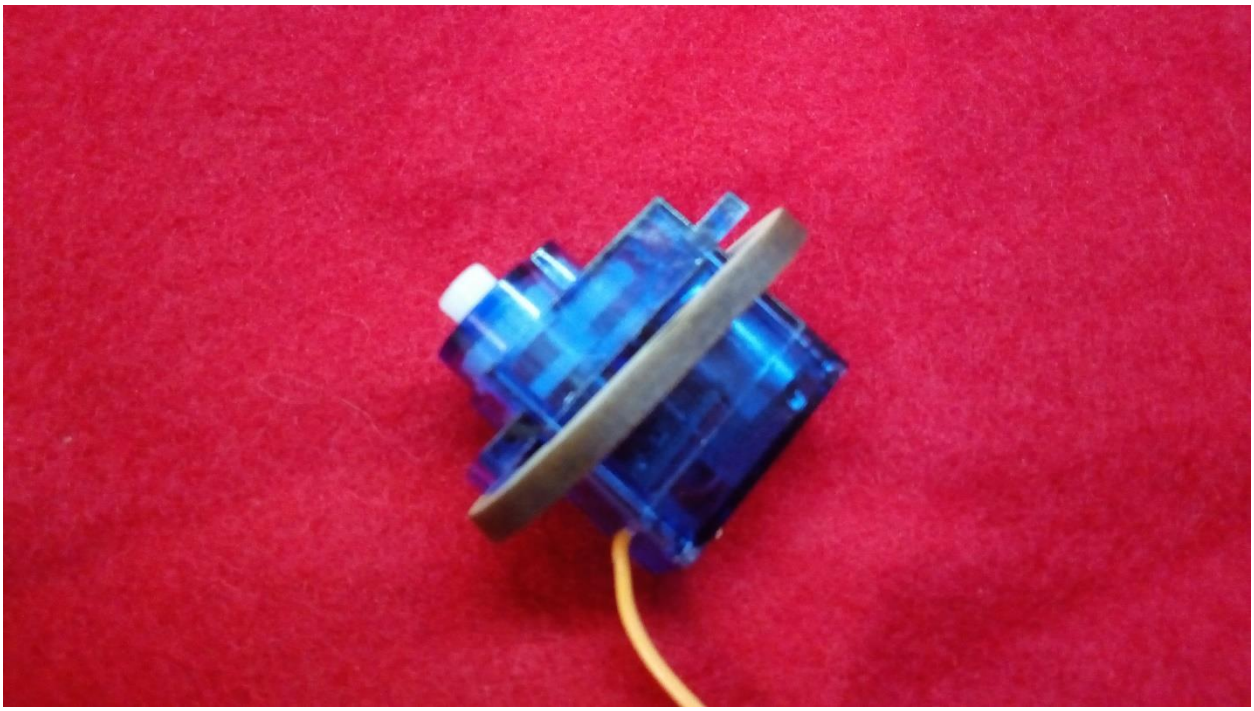
Lo siguiente será ponerle el collar al servo, esto se hará a lo largo de la construcción para sujetar los servomotores a este mismo.

Meteremos el cable a través del collar, se debe alinear la ranura del collar al extremo del servomotor donde sale el cable. Debe empujar el collar a la parte inferior del servomotor y estará listo (como se ve en las imágenes):



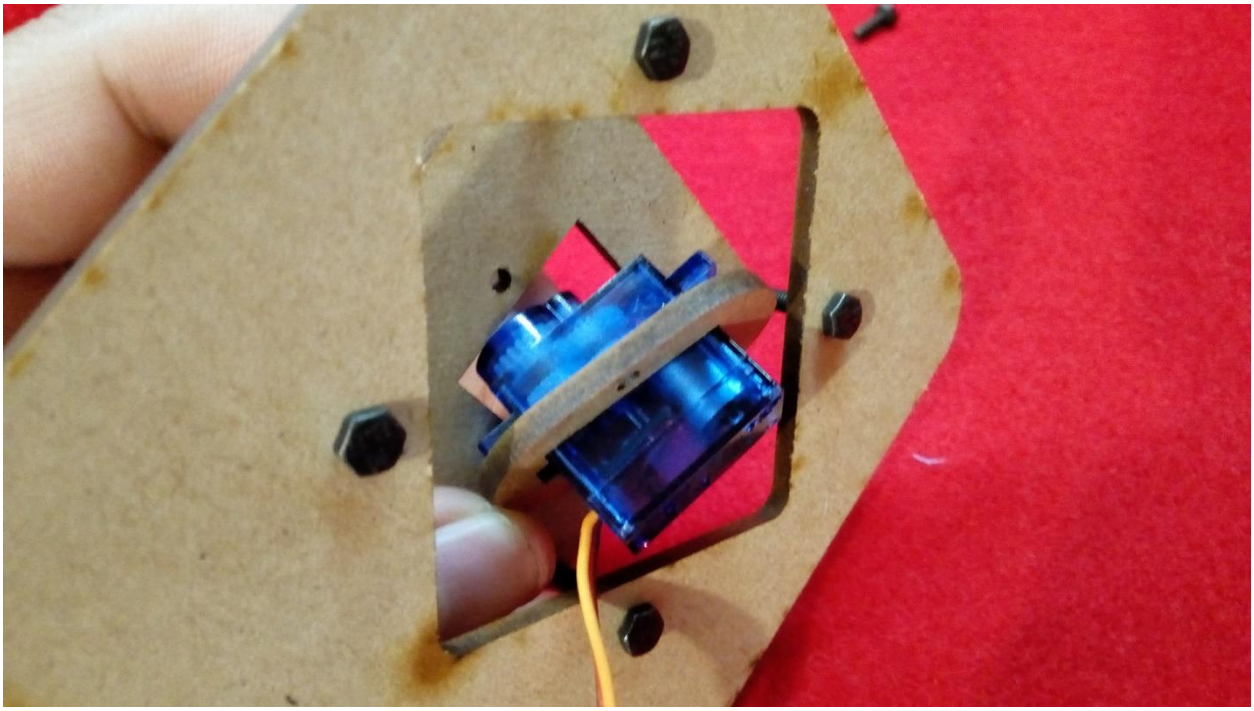


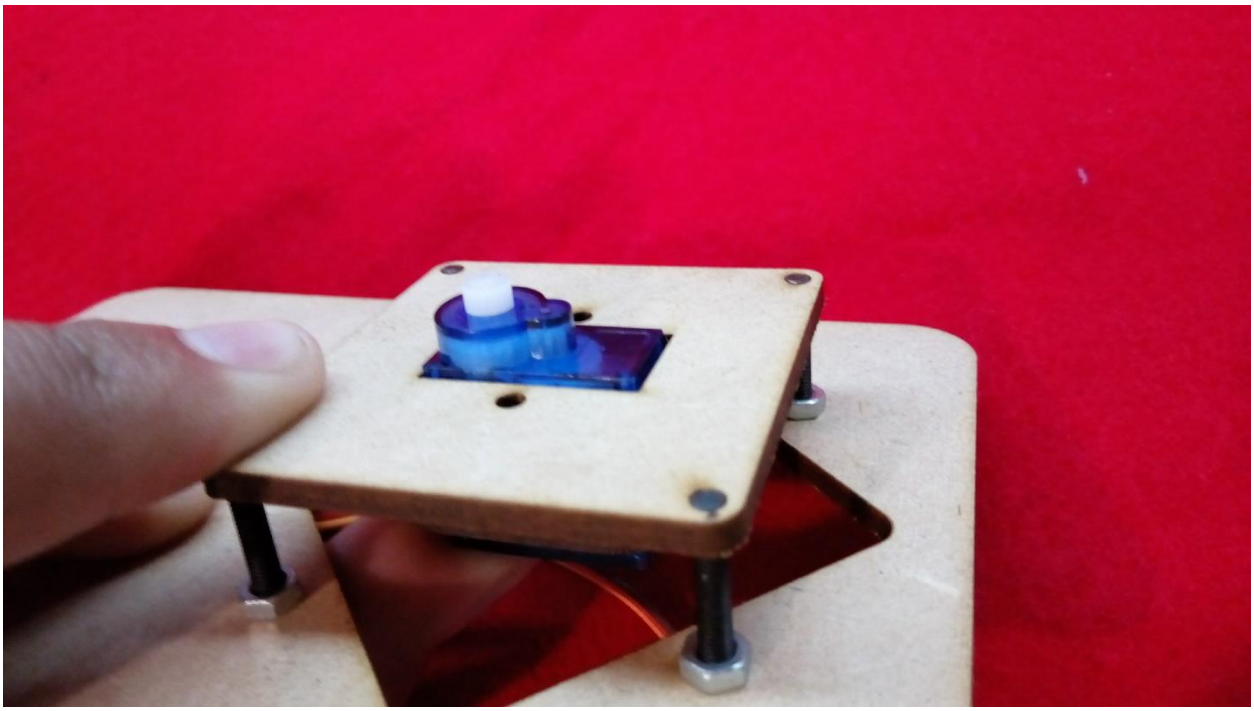


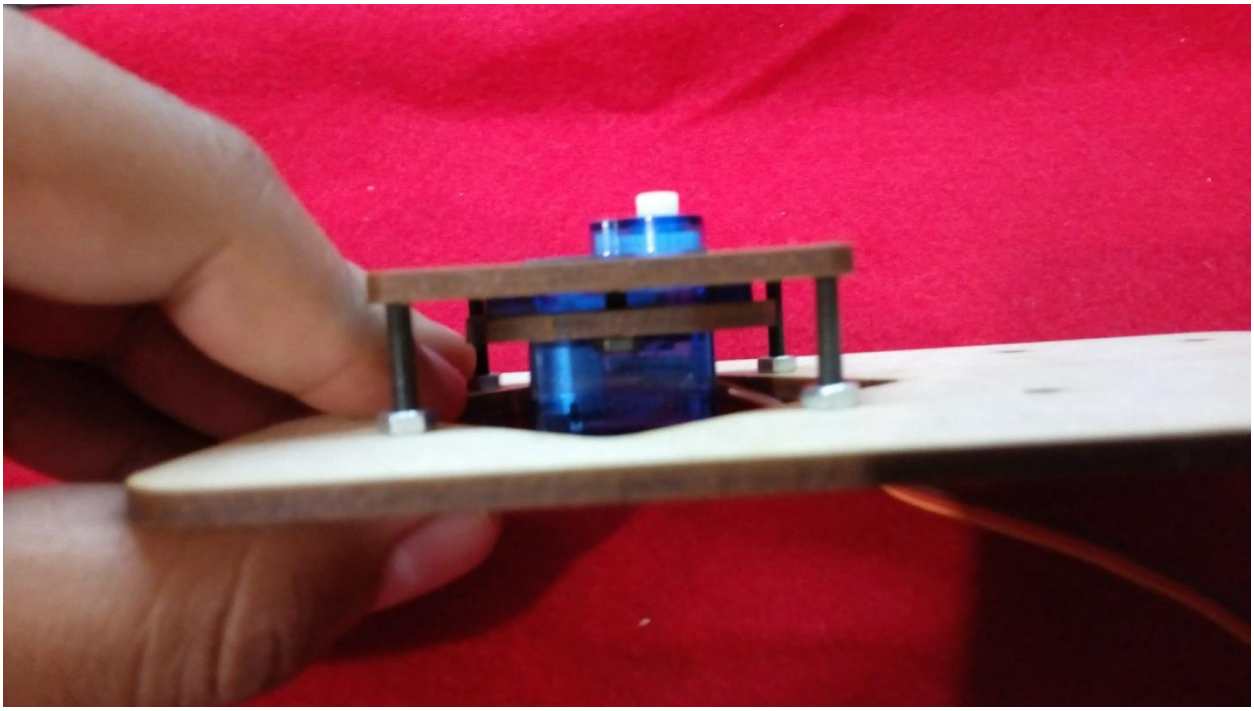


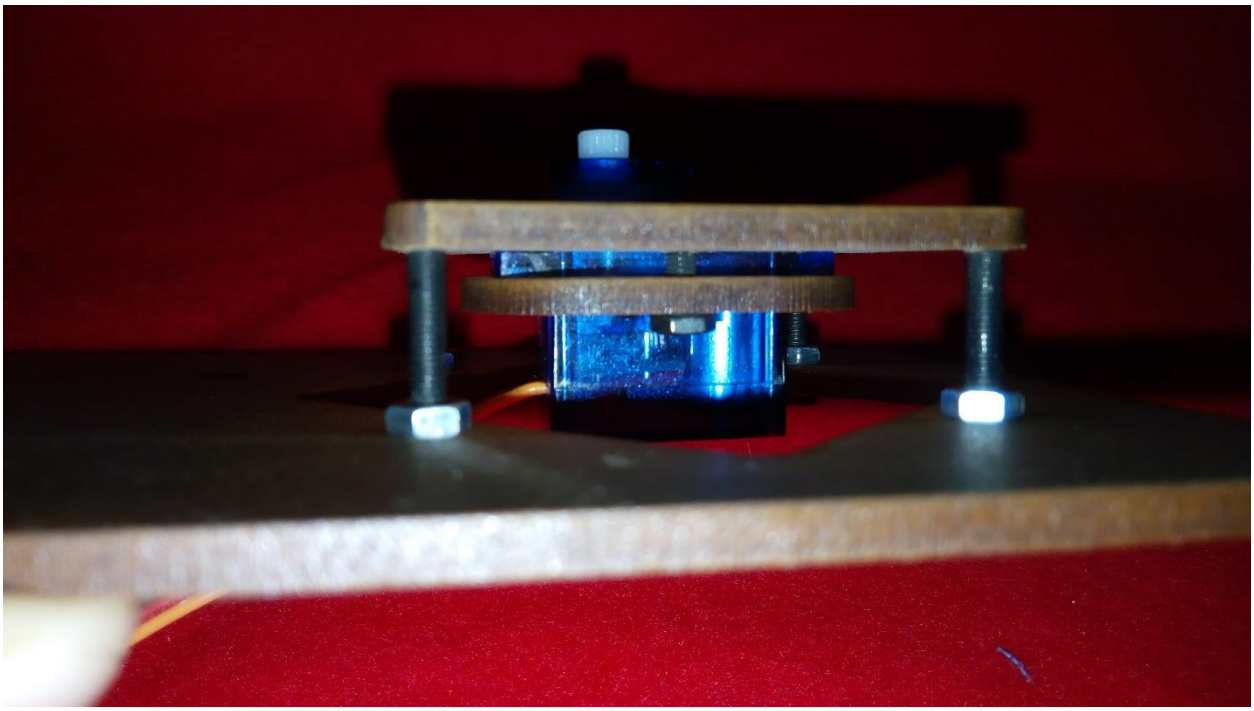
Ahora pase el servomotor a través del agujero cuadrado de la base, inserte los dos tornillos de 8mm por debajo de la base, de modo que pasen a través de los orificios del collar, apriete hasta que el servo se mantenga firme, NO APRETAR DEMASIADO.

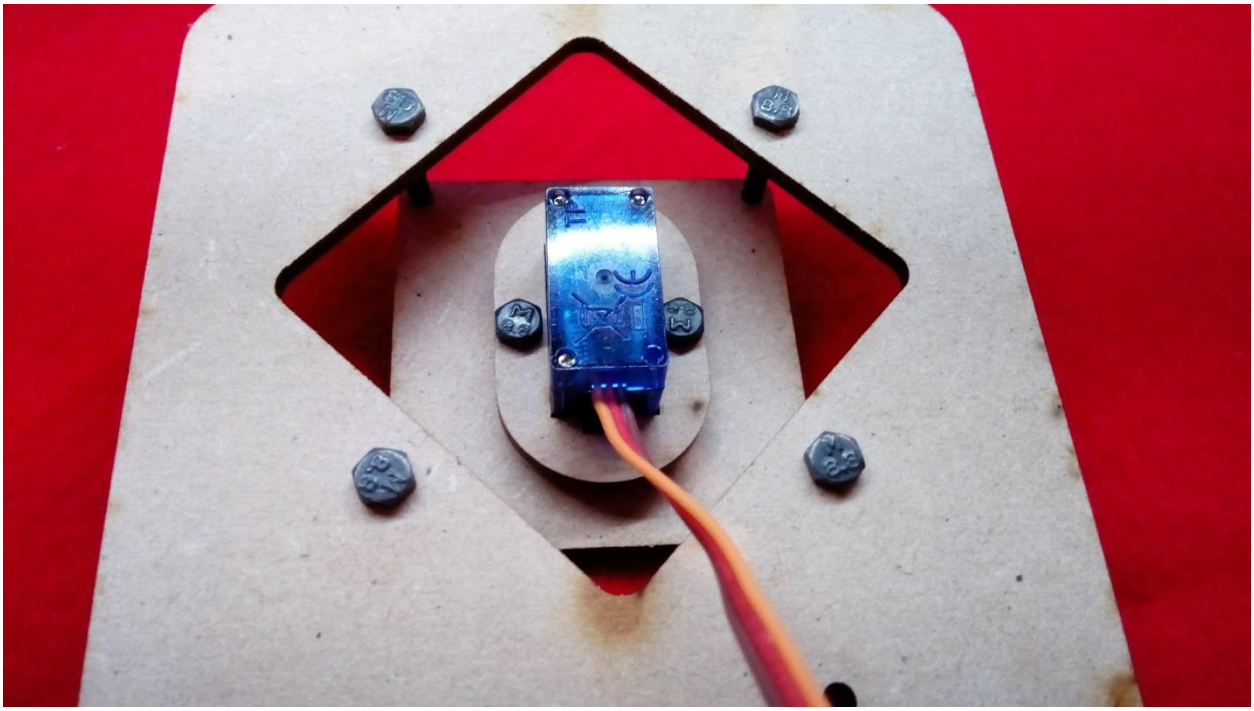
Nota: El servo debe de estar calibrado a 90 grados.









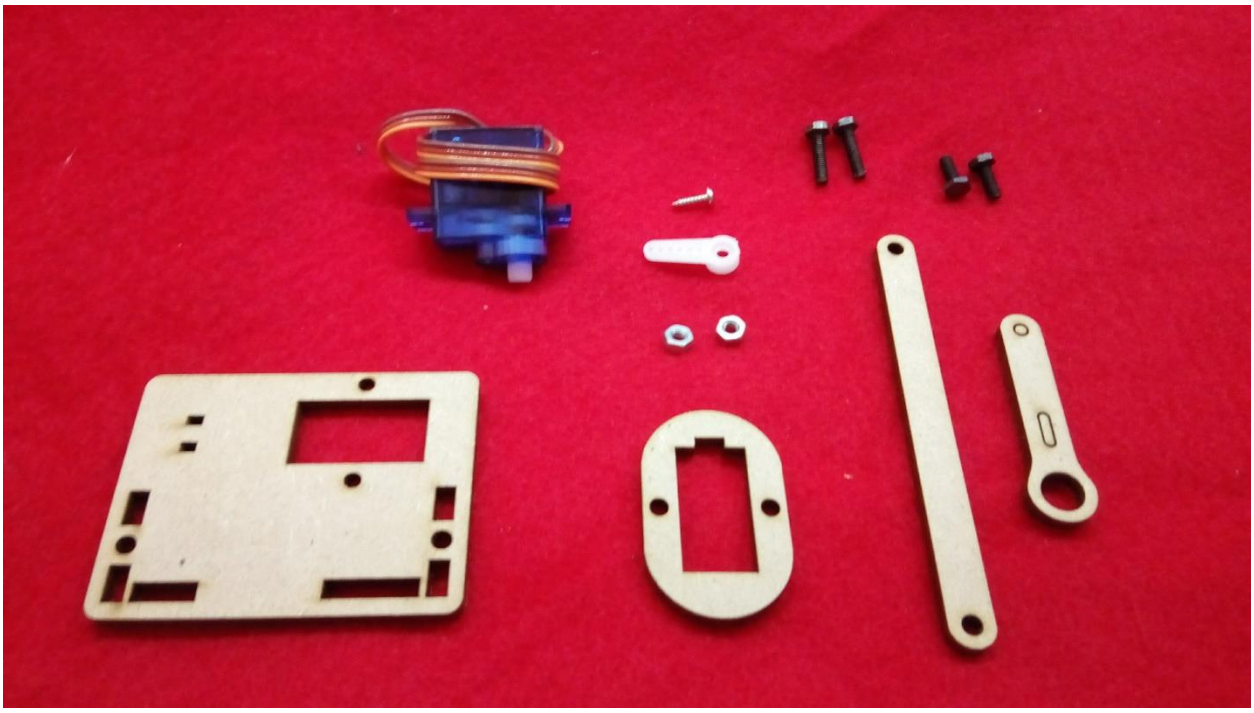


Muy bien, con eso hemos terminado la construcción de la base, ahora se construirá el lado izquierdo. A continuación se listan los materiales para hacerlo, recuerde seguir las instrucciones de armado al pie de la letra ya que si no lo hace puede que no funcione el brazo al finalizar el armado de este.

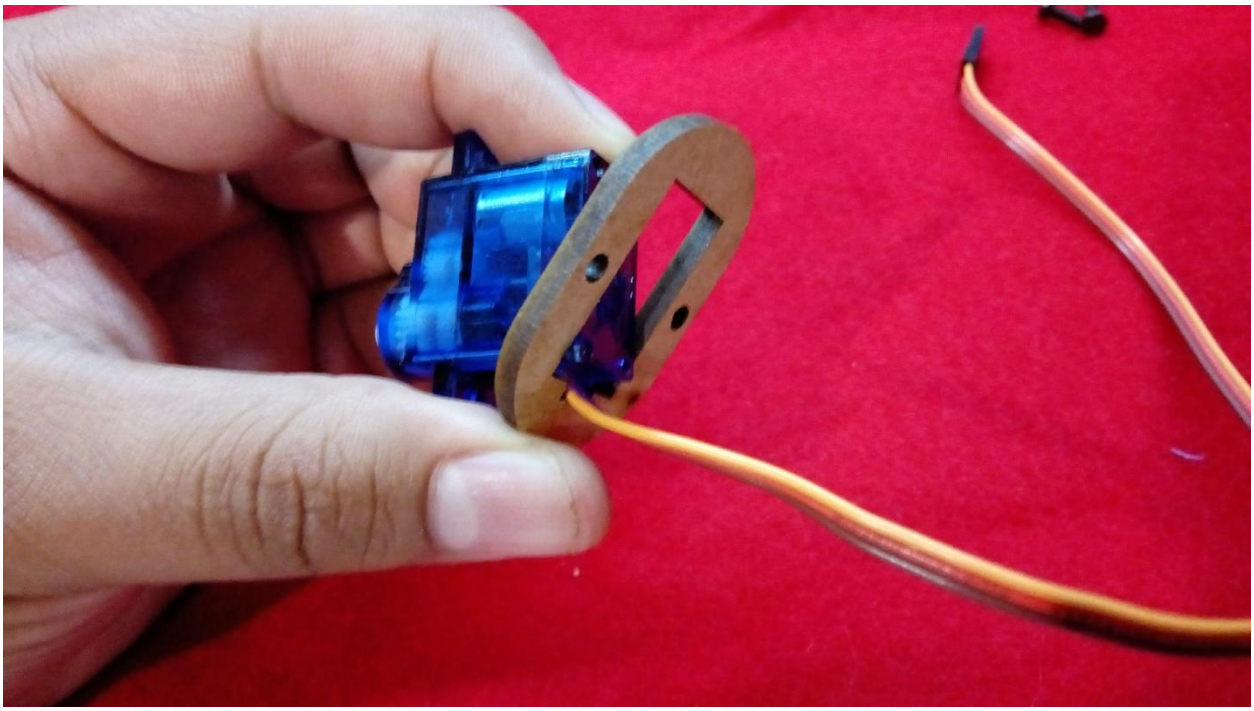
5.1.2 ARMADO DEL BRAZO – LADO IZQUIERDO

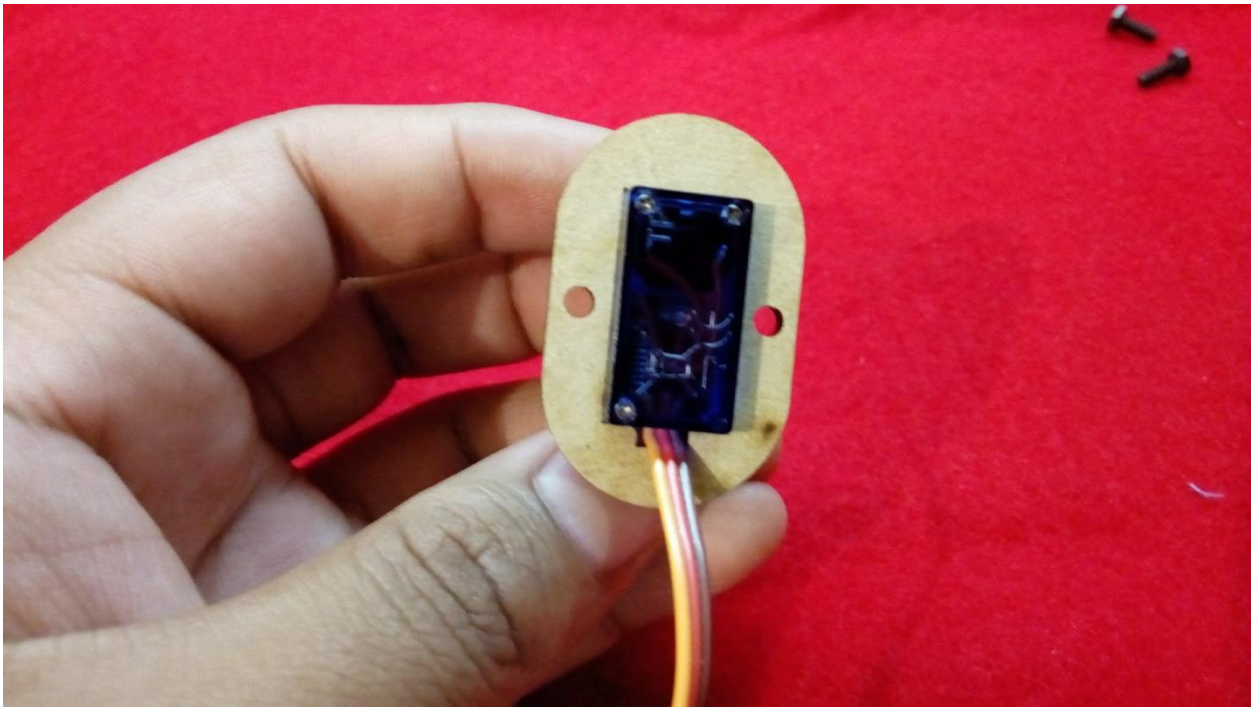
En esta parte se mostrará cómo armar el lado izquierdo del brazo, a continuación se lista el material necesario para la misma.

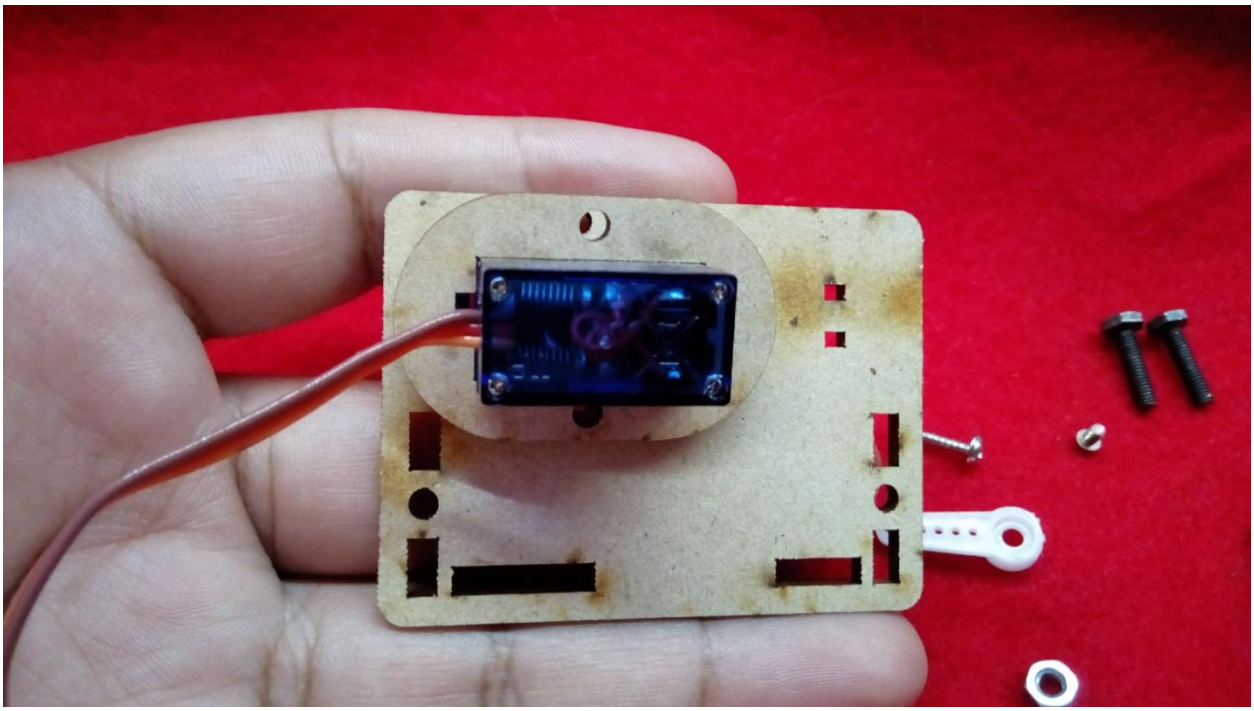
MATERIALES
2 Tornillos de 8mm
2 Tornillos de 12mm
2 Tuercas
1 Tornillo de 6mm
1 Collar
1 Servomotor
1 Rectángulo como el de la imagen
1 Placa recta redondeada como en la imagen
1 Placa corta con agujero, como en la imagen
1 Tornillo de servo largo
1 Tornillo de servo corto

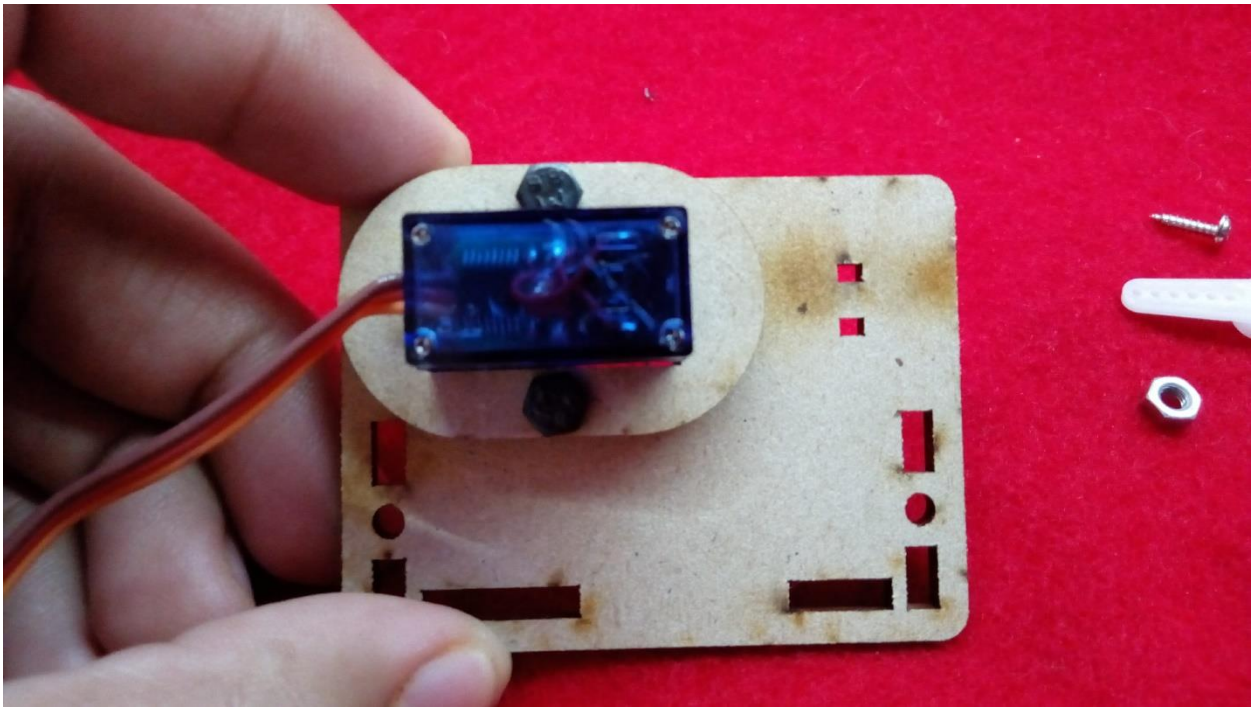


Monte el collar como se hizo anteriormente con el servomotor y atornille la pieza con los dos tornillos de 8mm (ver imagen), recuerde no apretar mucho. Preste mucha atención a la orientación en la que se pone el servo. Observe la dirección del cable.

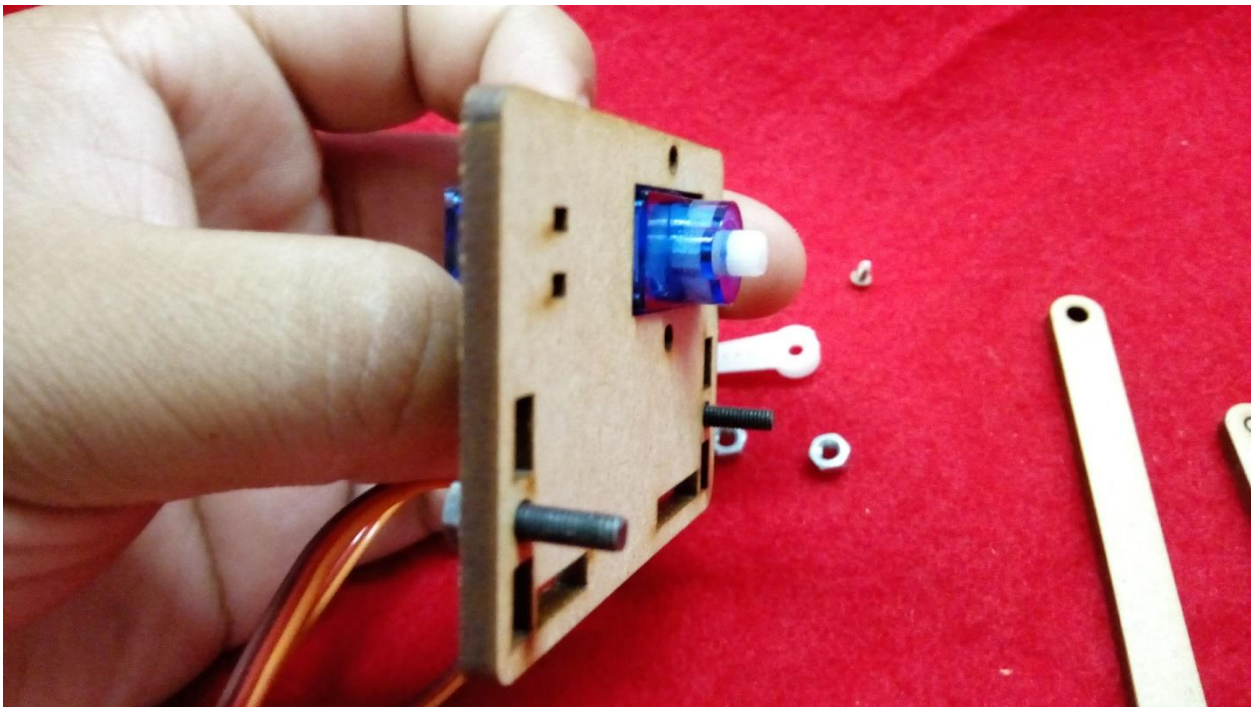


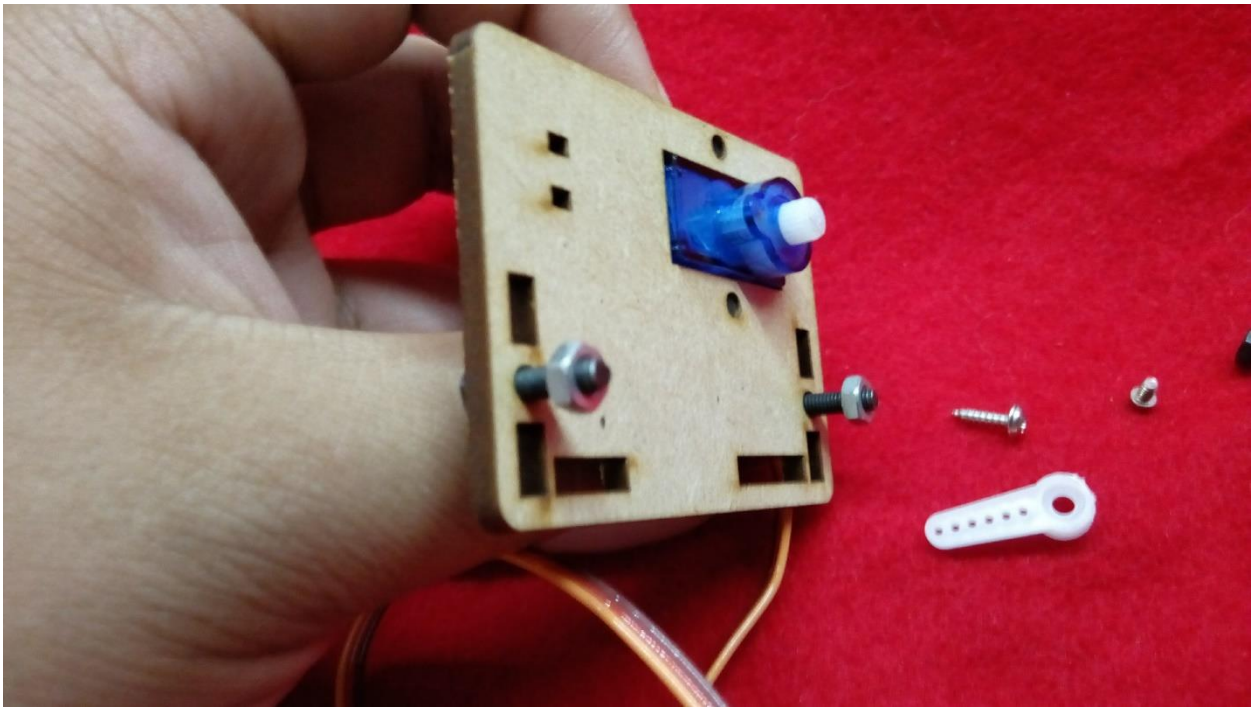




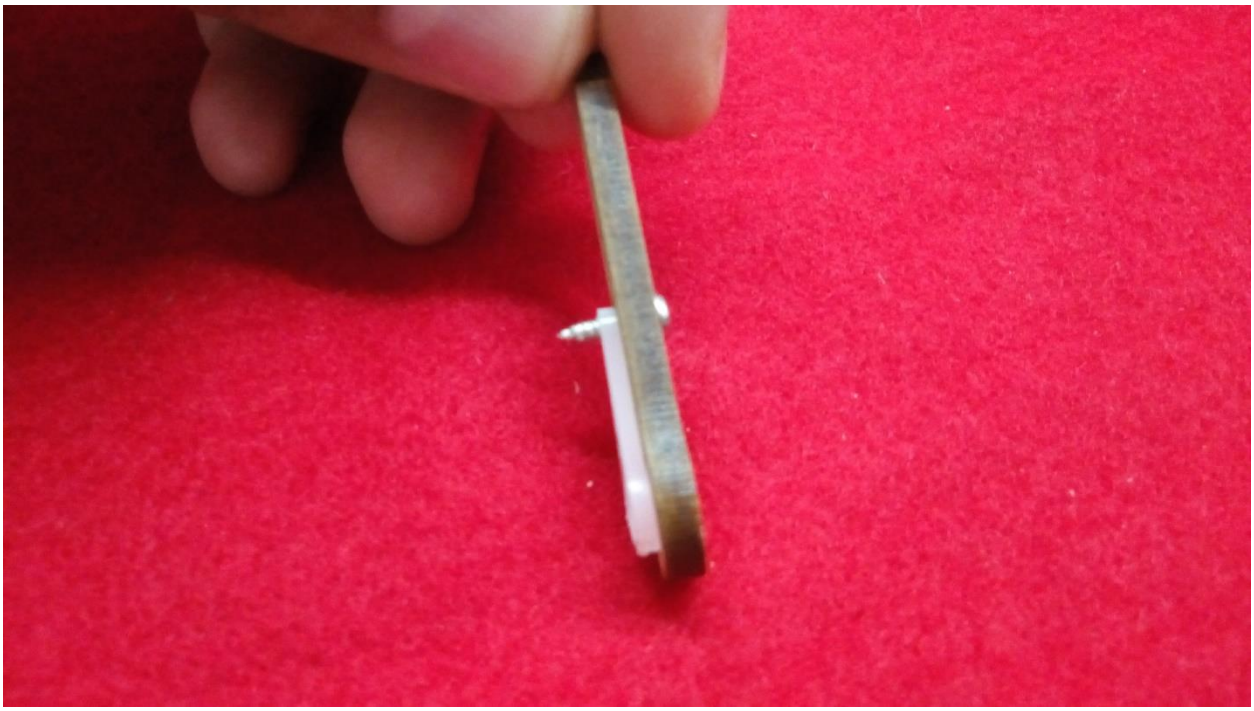


Ahora inserte los dos tornillos de 12mm a través de los agujeros redondos y coloque las tuercas a la mitad.



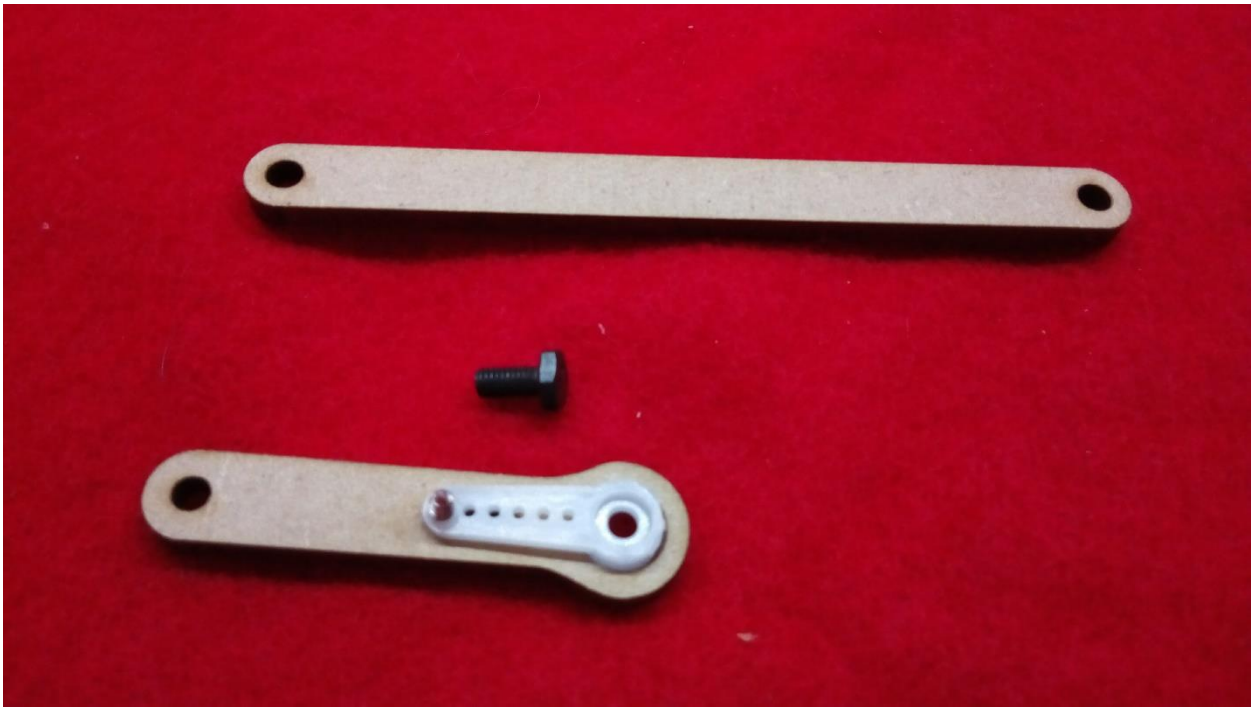


Ahora construiremos la palanca, conecte la pieza blanca del servomotor a la parte de la palanca, sujete usando el tornillo largo del servomotor, se asomará en la parte posterior (ver imágenes).

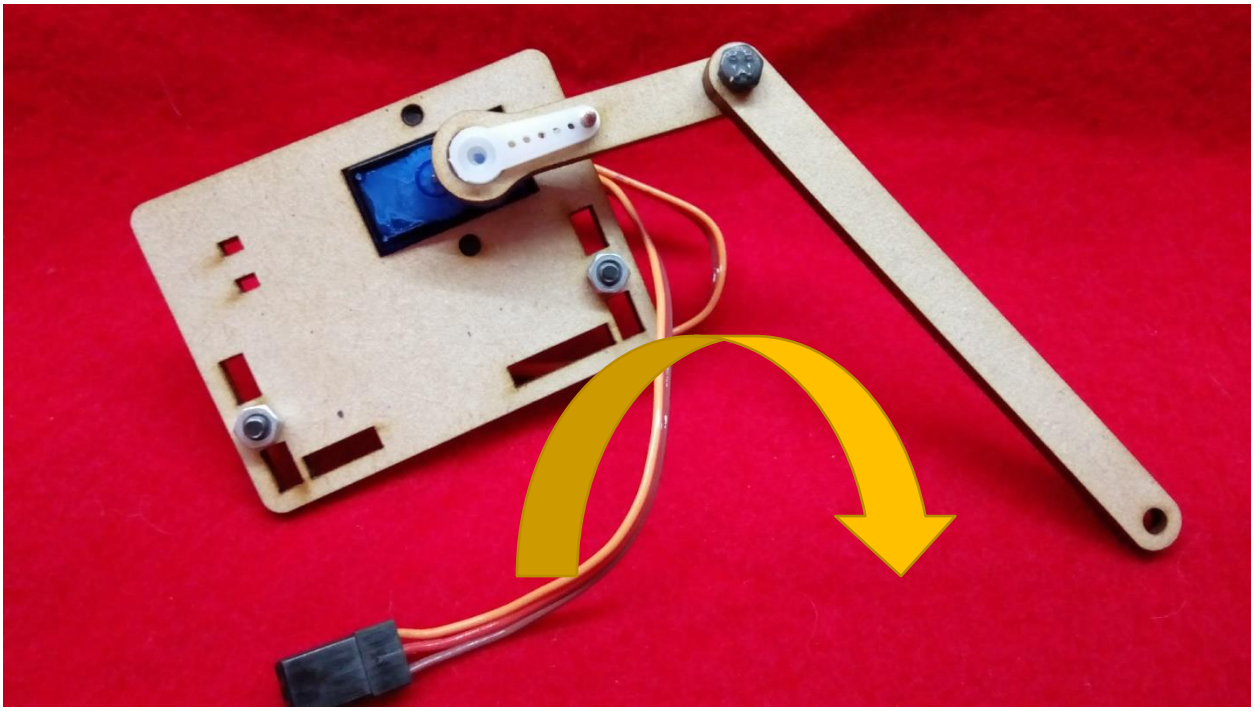


Ahora conecte la palanca larga a la palanca del servo con el tornillo de 6mm. Esta es la primera parte móvil, no ajustar demasiado permitiendo que quede un poco floja.

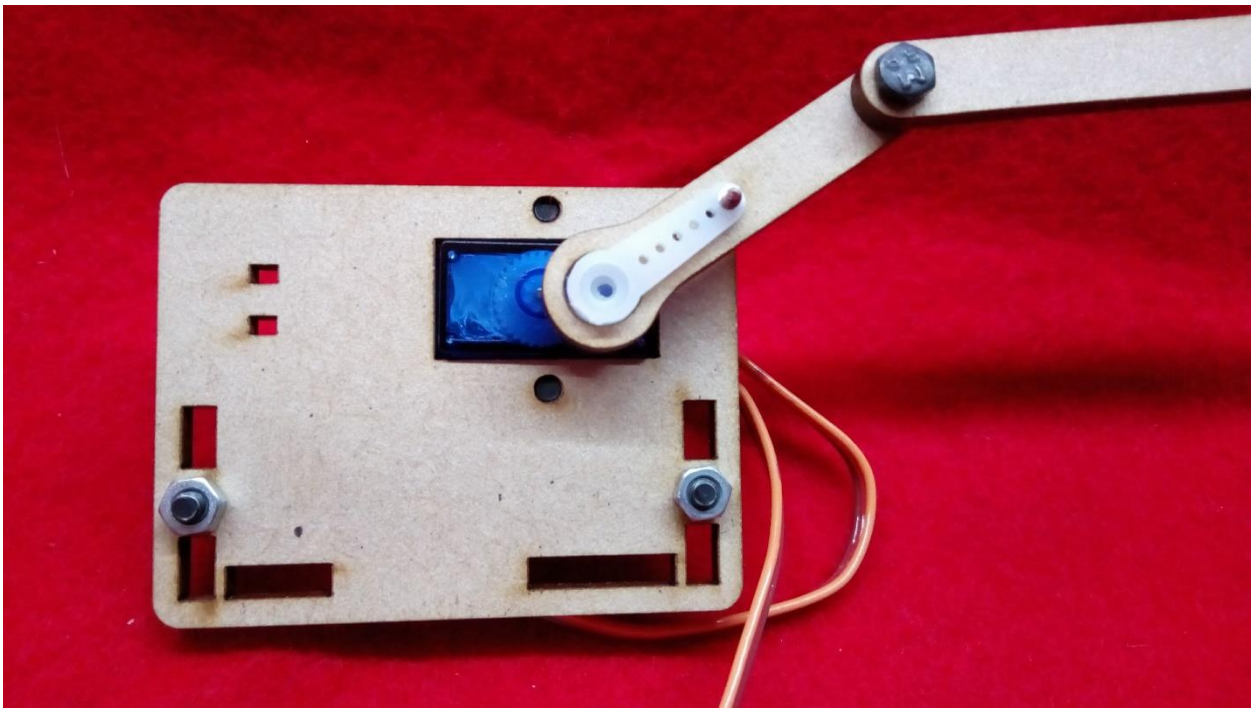
Tenga en cuenta esto para las próximas partes movibles.



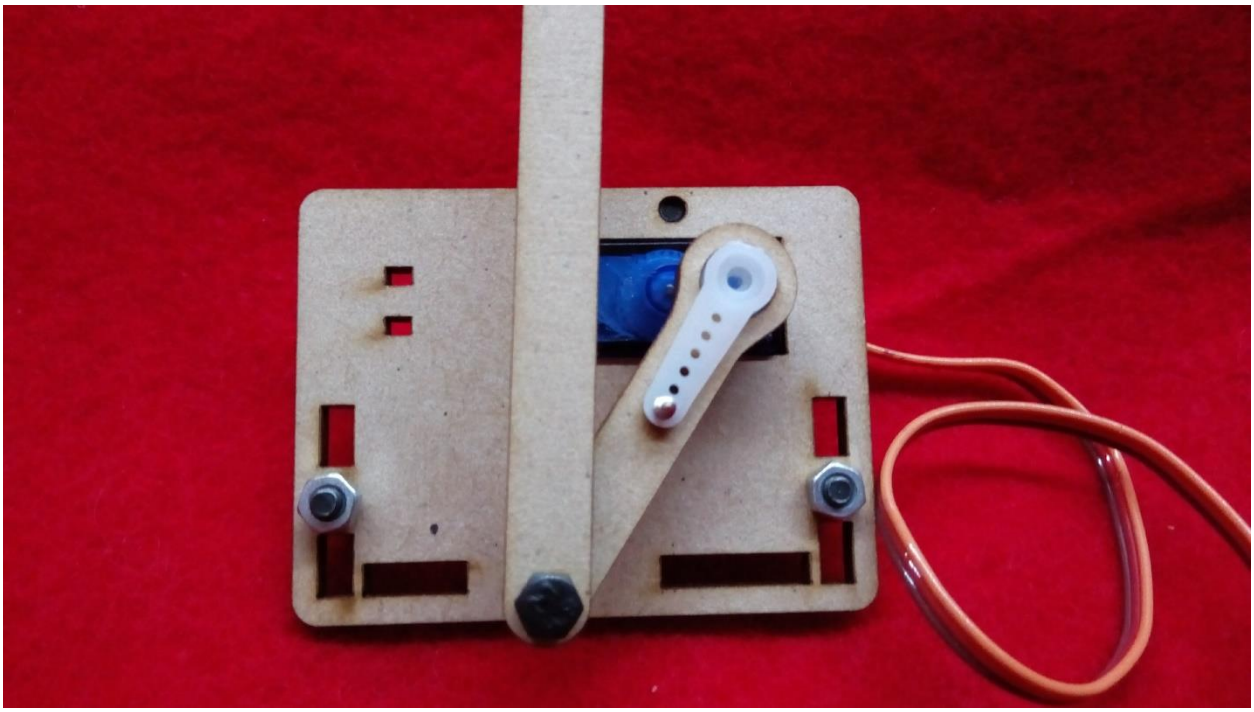
Esta parte es muy importante ya que se conectará la palanca y encontraremos los límites de movimiento. Conecte la palanca que se acaba de implementar al servomotor, simplemente se debe empujar, los servomotores se pueden girar a mano (no lo giren muy fuerte ni lo fuercen) así que suavemente gire todo el camino en sentido a las agujas del reloj hasta que se detenga (como se ve en la imagen).



Cuando se detenga, retire la palanca y colóquela de modo que coincida con la siguiente imagen:



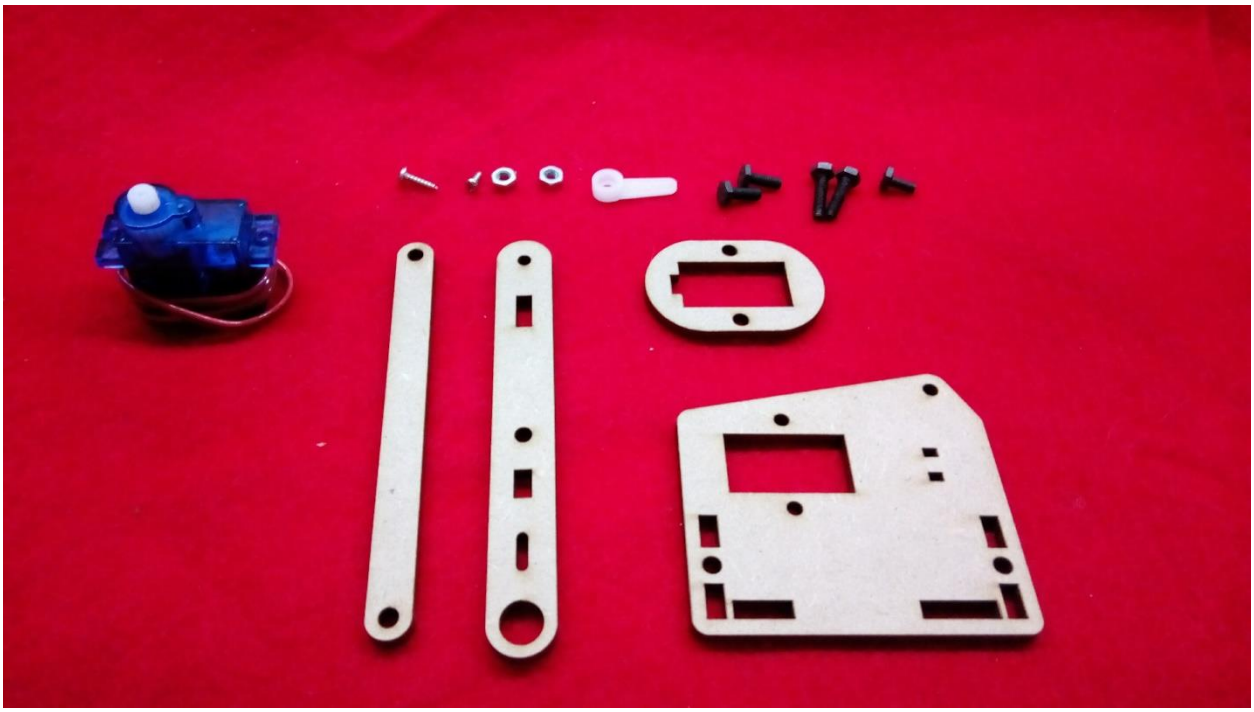
Coloque el tornillo pequeño para sujetar la palanca al servomotor y apriete un poco, cuando haya hecho eso gire el servomotor con la palanca, debe de hacer el recorrido en sentido contrario a las agujas del reloj, al final debe quedar como en la imagen:



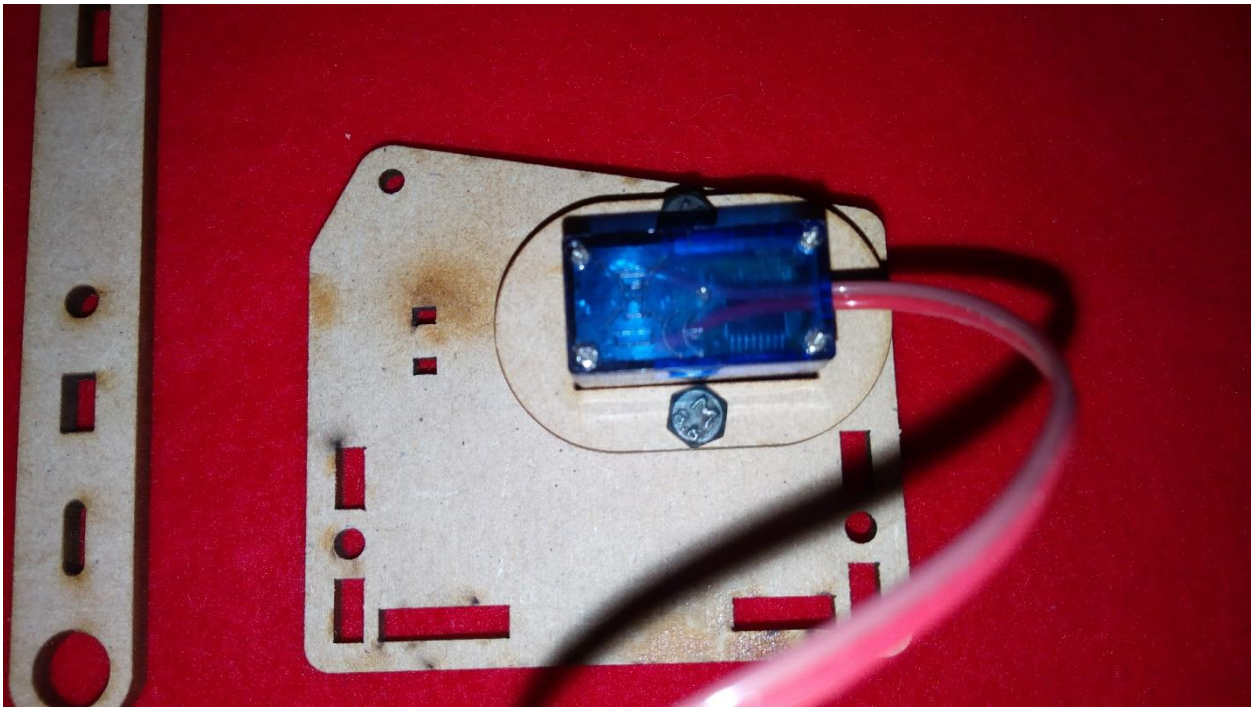
5.1.3 ARMADO DEL BRAZO – LADO DERECHO

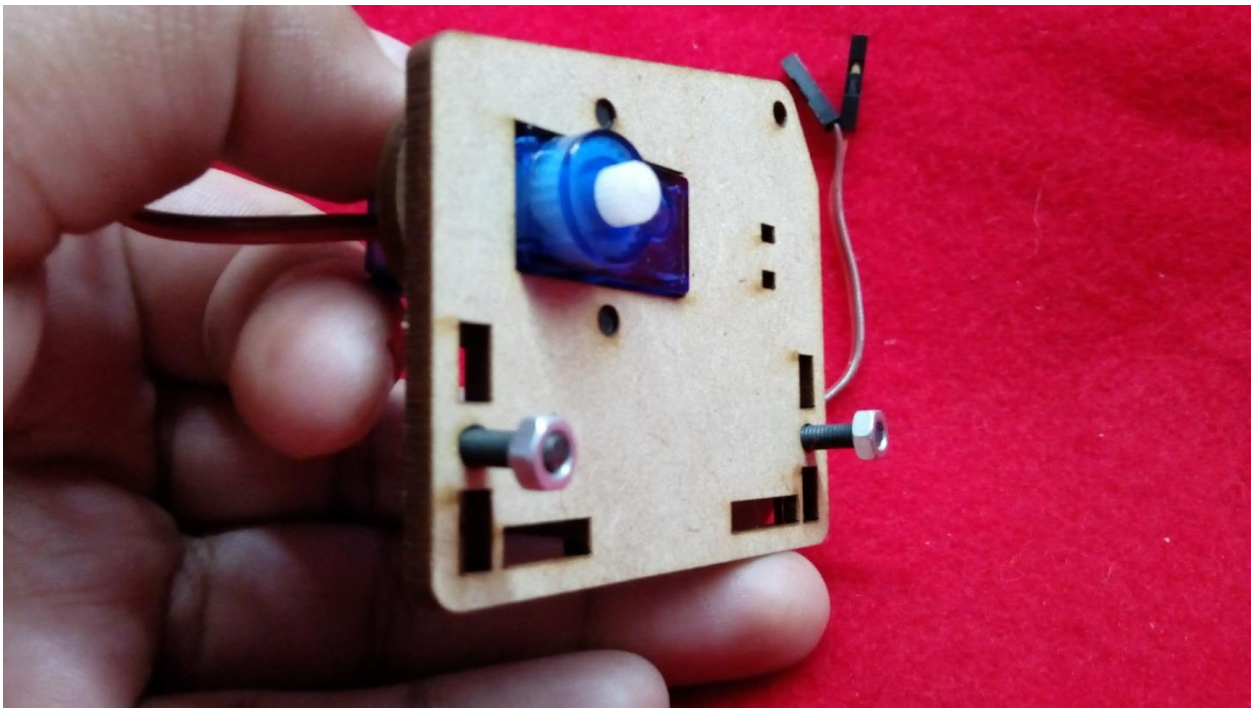
En esta parte se mostrará cómo armar el lado derecho del brazo, a continuación se lista el material necesario para el mismo.

MATERIALES
2 Tornillos de 8mm
2 Tornillos de 12mm
2 Tuercas
1 Tornillo de 6mm
1 Collar
1 Servomotor
1 Rectángulo como el de la imagen
1 Placa recta redondeada como en la imagen
1 Placa corta con agujero, como en la imagen
1 Tornillo de servo largo
1 Tornillo de servo corto
1 Pieza plástica del servomotor (ver imagen)
1 Pieza lateral derecha (ver imagen)
1 Palanca larga (ver imagen)
1 Palanca central (ver imagen)

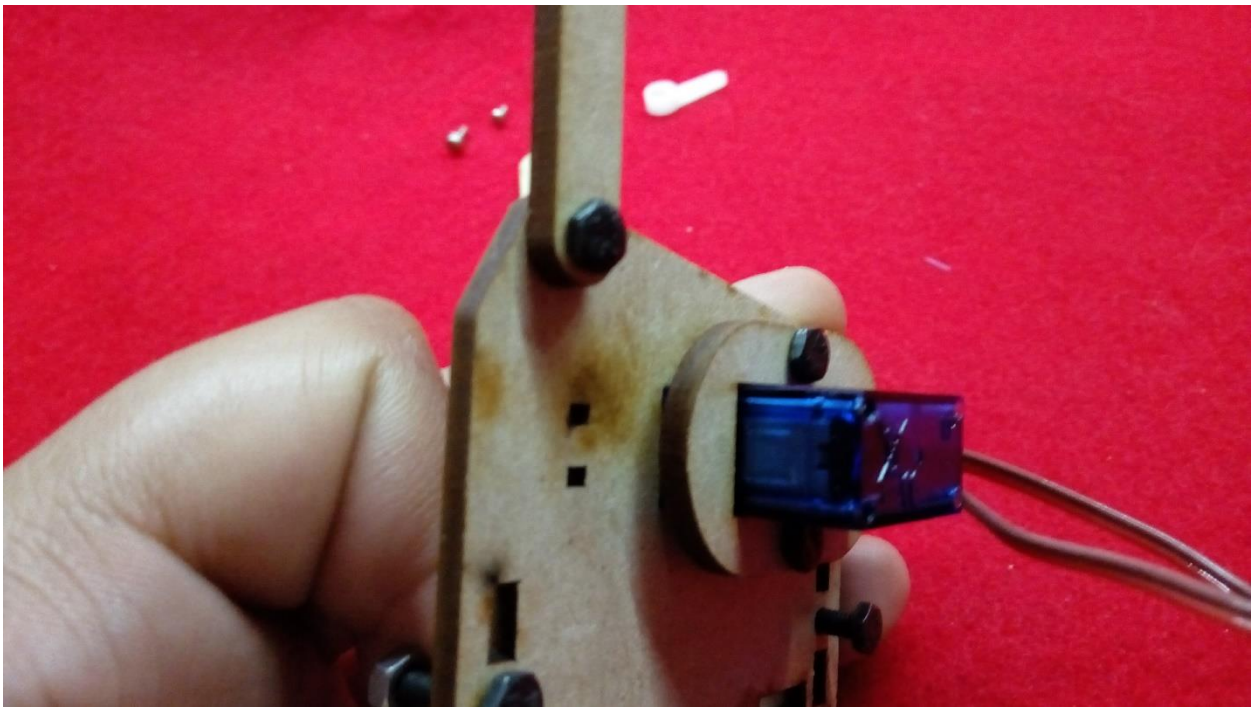


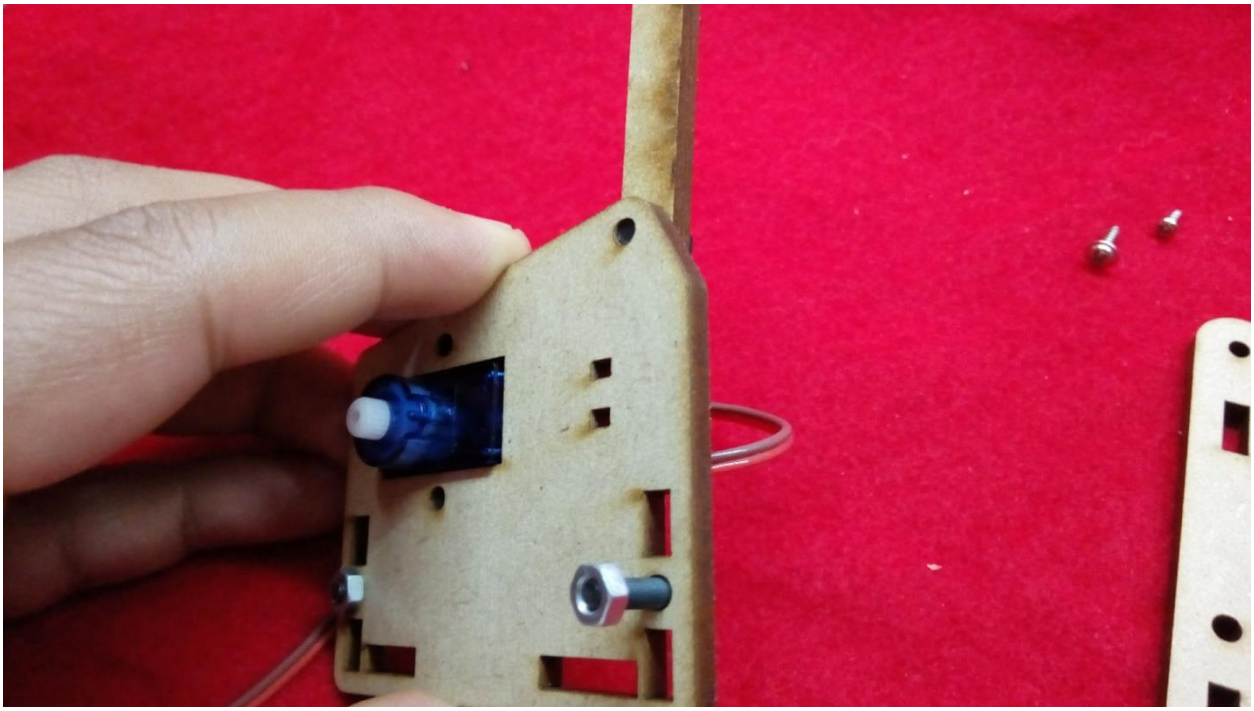
Ahora debe ponerle el collar al servomotor, montarlo sobre la pieza y sujetarlo con los tornillos de 8mm, después debe insertar los tronillos de 12mm con las respectivas tuercas (ver imágenes).



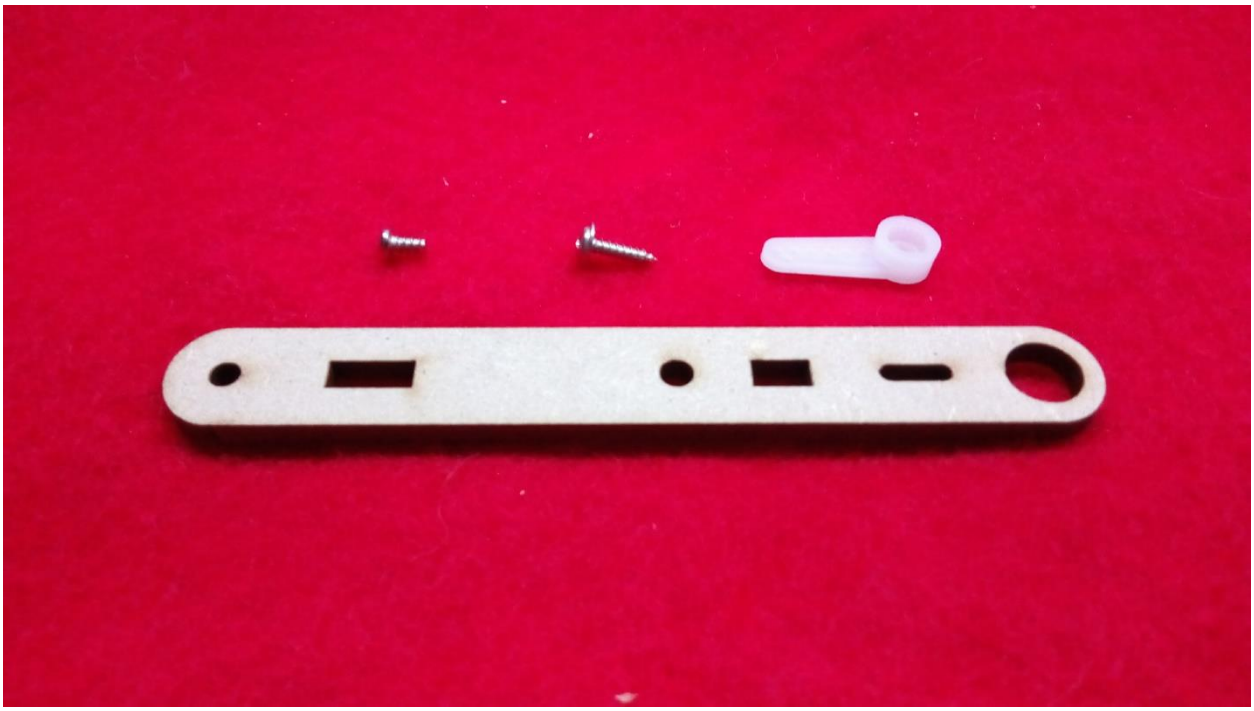


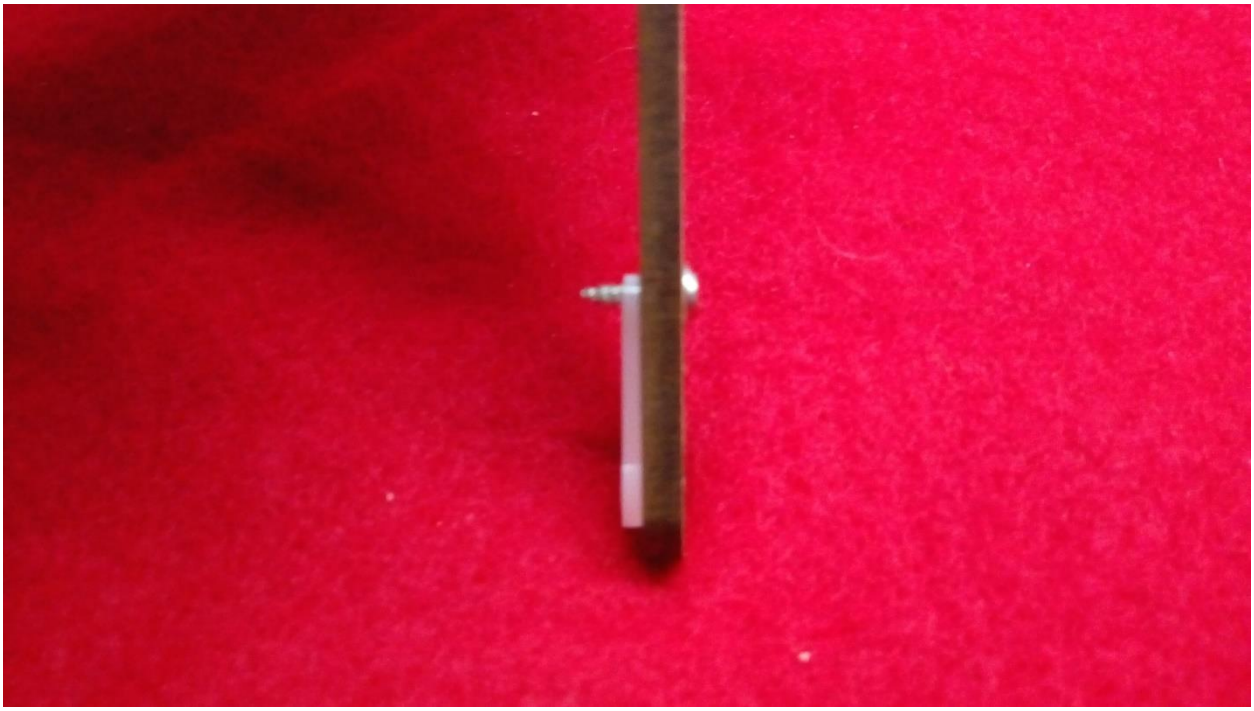
Conecte la palanca larga al exterior de la pieza lateral derecha con el tornillo de 6mm, esta será la otra parte móvil, recuerde no apretar mucho.



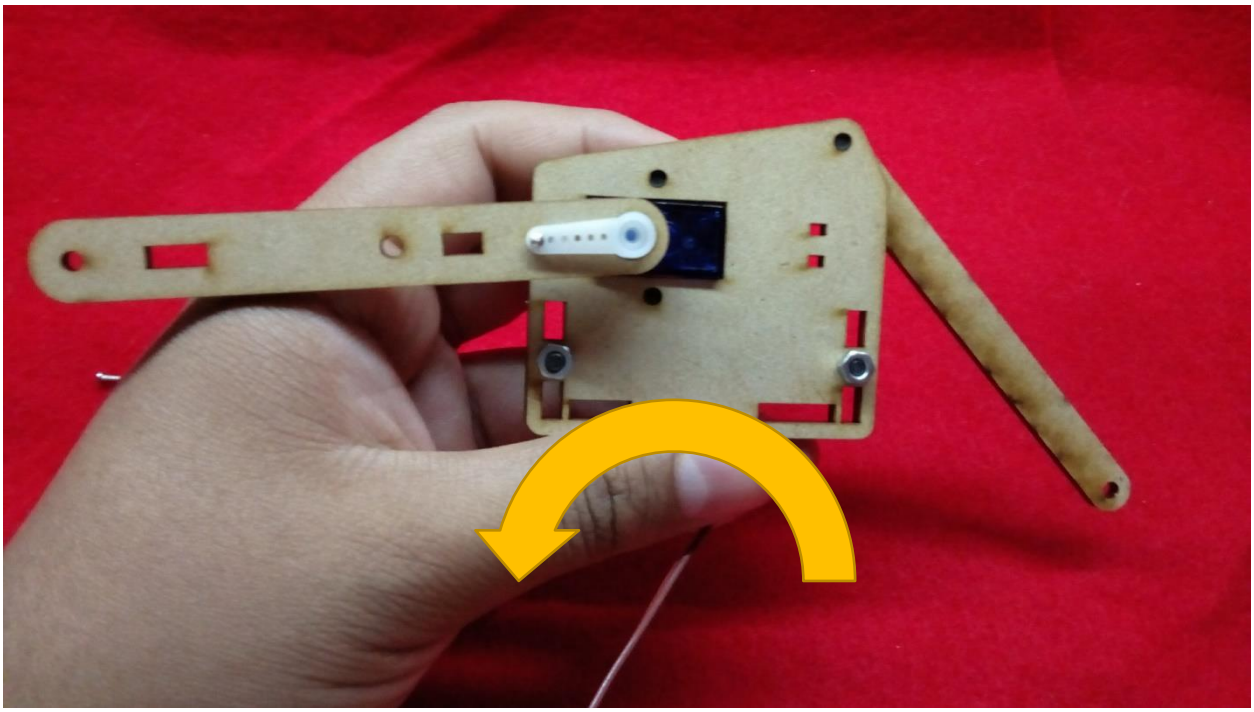


Una vez hecho esto se debe de conectar la palanca y establecer límites, debe montar la pieza blanca del servomotor en la palanca larga, esta debe de sujetarse con el tornillo largo del servomotor (ver imágenes).

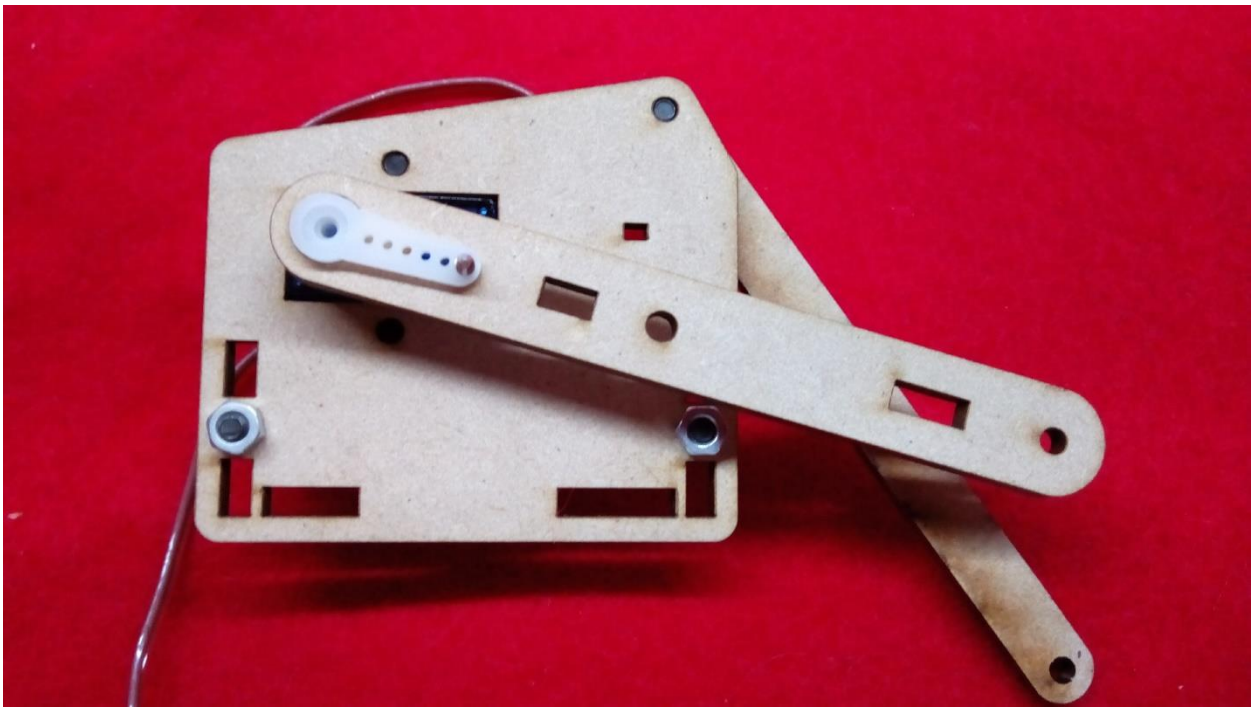




Ahora se montará la palanca en el servomotor, empuje sobre el servomotor y gire suavemente todo el camino hacia la izquierda, retire la palanca y vuelva a colocarla como se ve en las siguientes imágenes:



Inserte el tornillo pequeño, sujete la palanca al servomotor y gire en sentido de las agujas del reloj, debe de coincidir con la siguiente imagen.



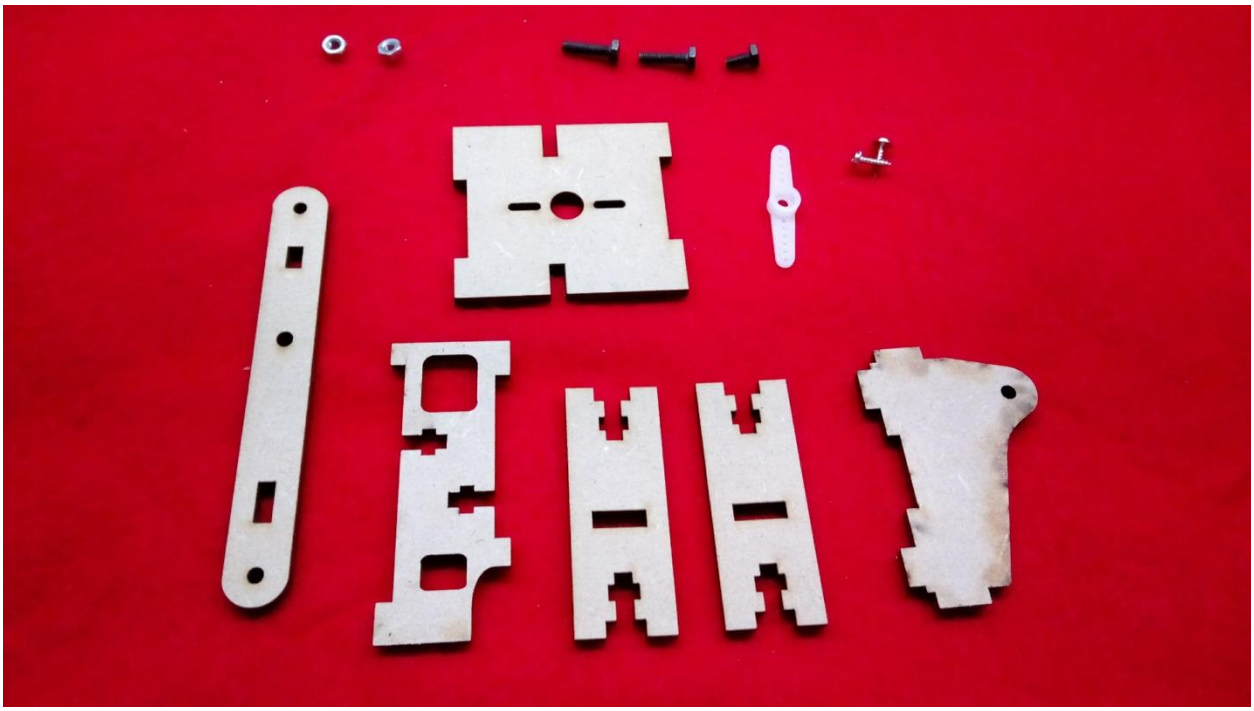
5.1.4 ARMADO DEL BRAZO – PARTE CENTRAL Y EL “CERDO”

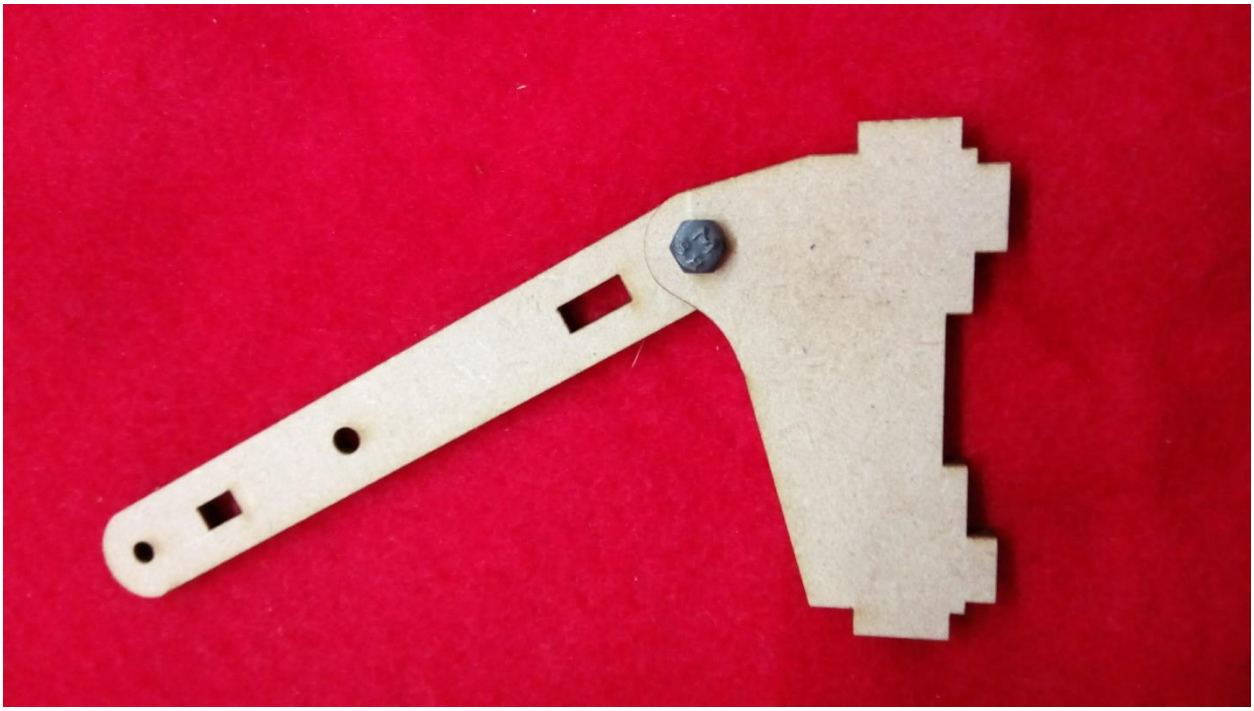
En esta parte se mostrará cómo armar la parte central del brazo con el “cerdo”, a continuación se lista el material necesario para la misma.

MATERIALES
2 Tornillos de 12mm
1 Tornillo de 6mm
2 Tuercas
2 Tornillos de servomotor largo
1 Tornillo de servomotor corto
1 Pieza larga del servo (ver imagen)

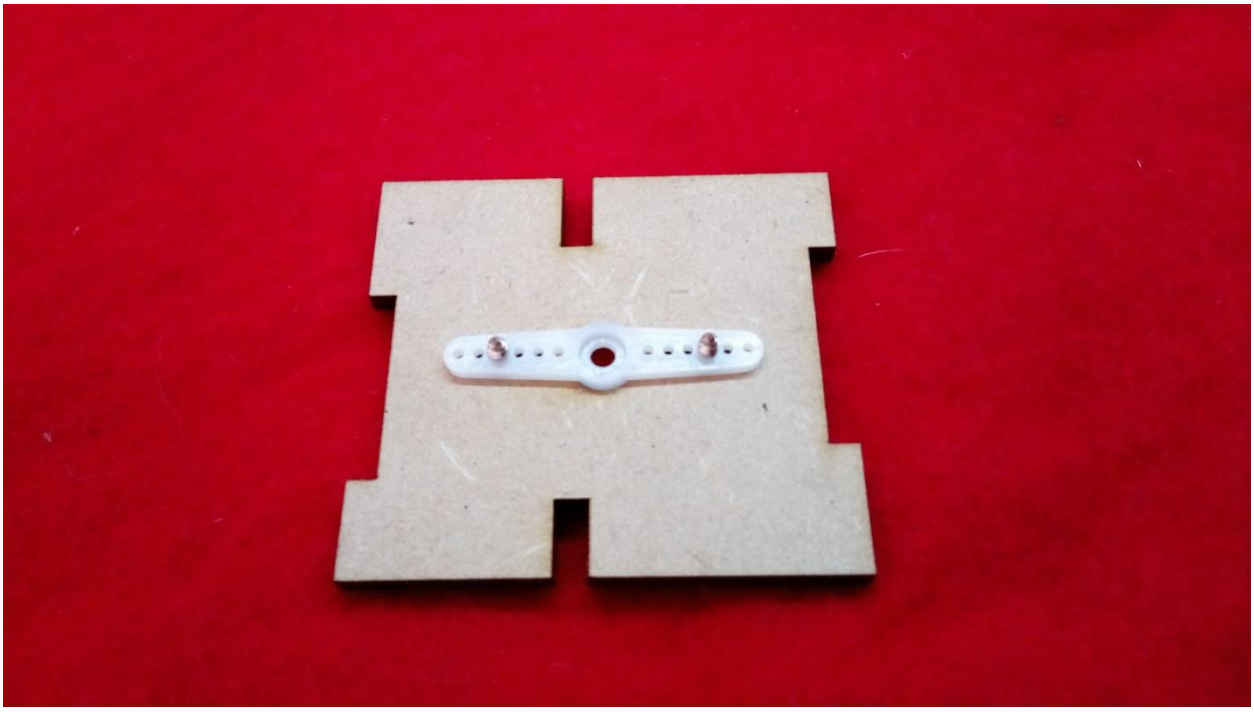
Ahora se deben de unir los dos lados que se han construido con la parte central y conocer una de las piezas llamada “cerdo”. El cerdo se muestra en la imagen y debe de unirse con la palanca larga.

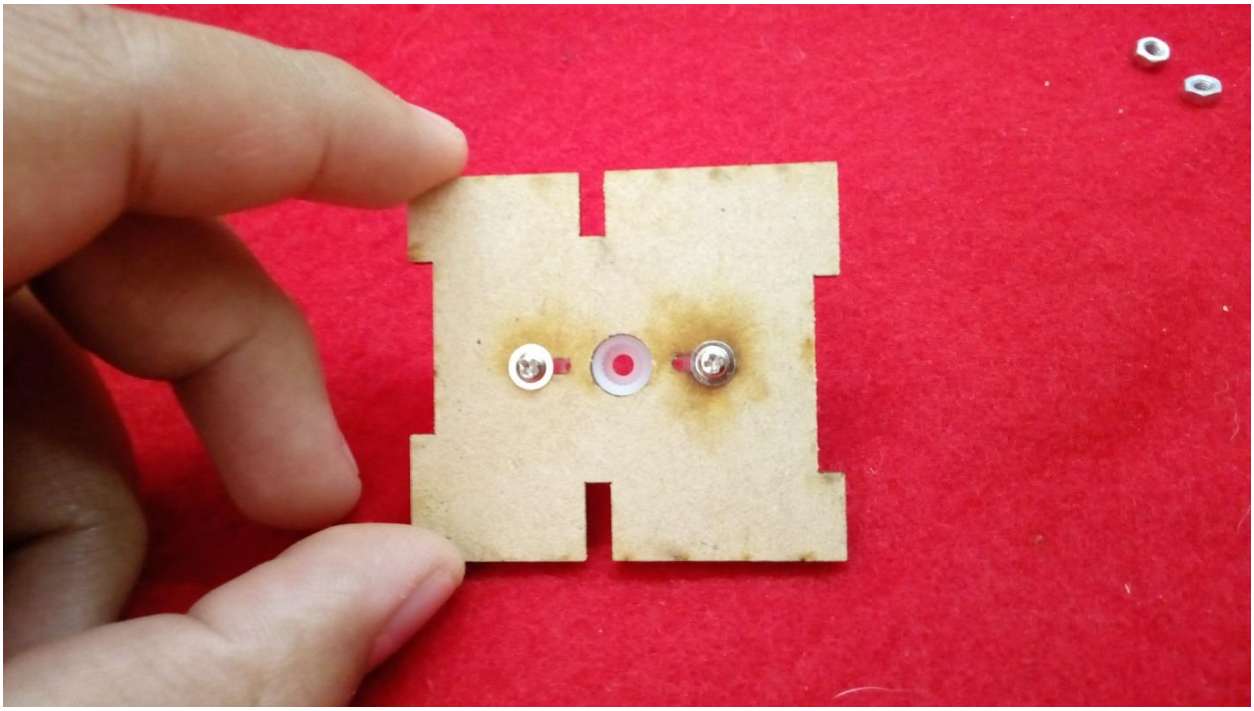
Conecte el “cerdo” a la palanca central con el tronillo de 6mm y como siempre note la orientación de este tornillo, recuerde no apretar mucho.



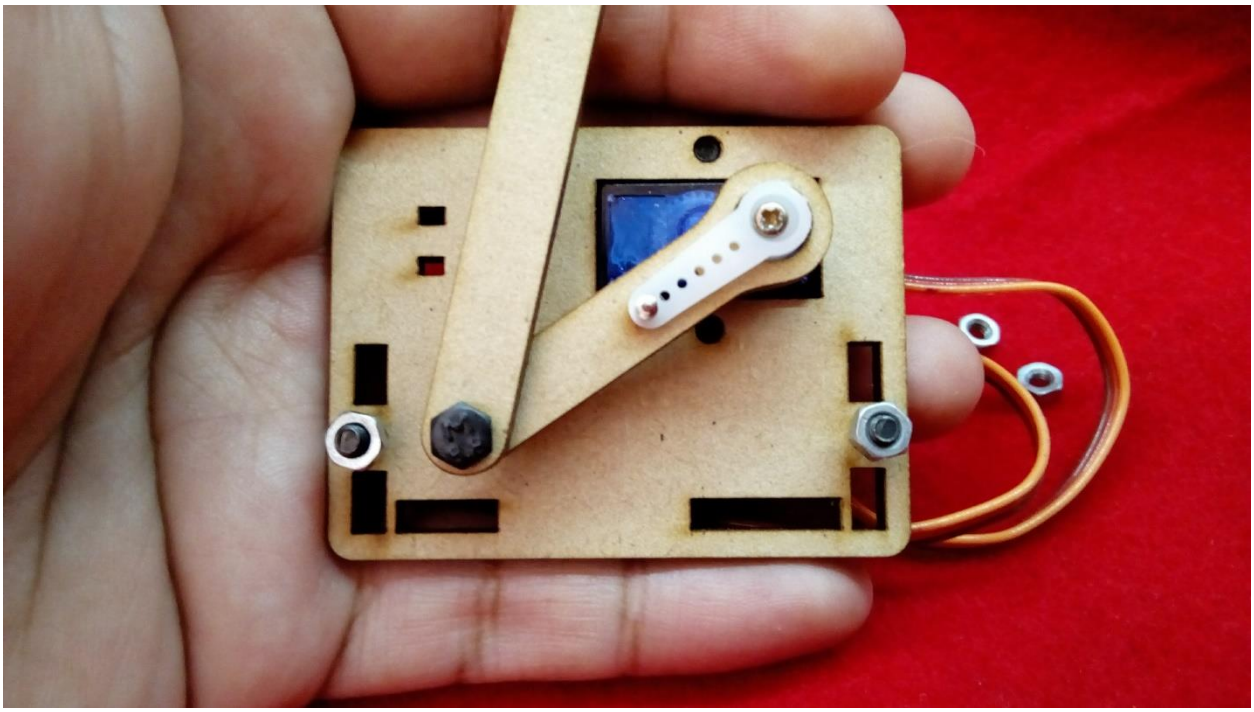


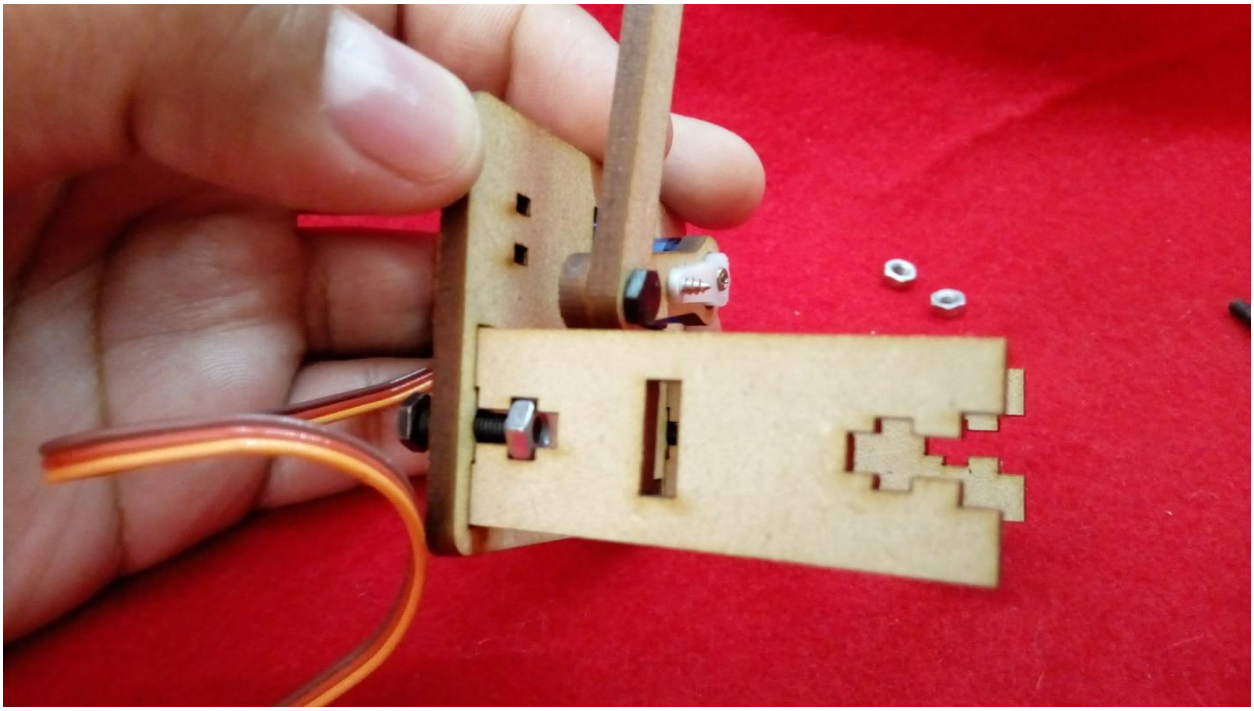
Fije la pieza del servomotor a la pieza de la base, ahora utilice los dos tronillos largos del servomotor para sujetar la pieza de base cuadrada (ver imagen).

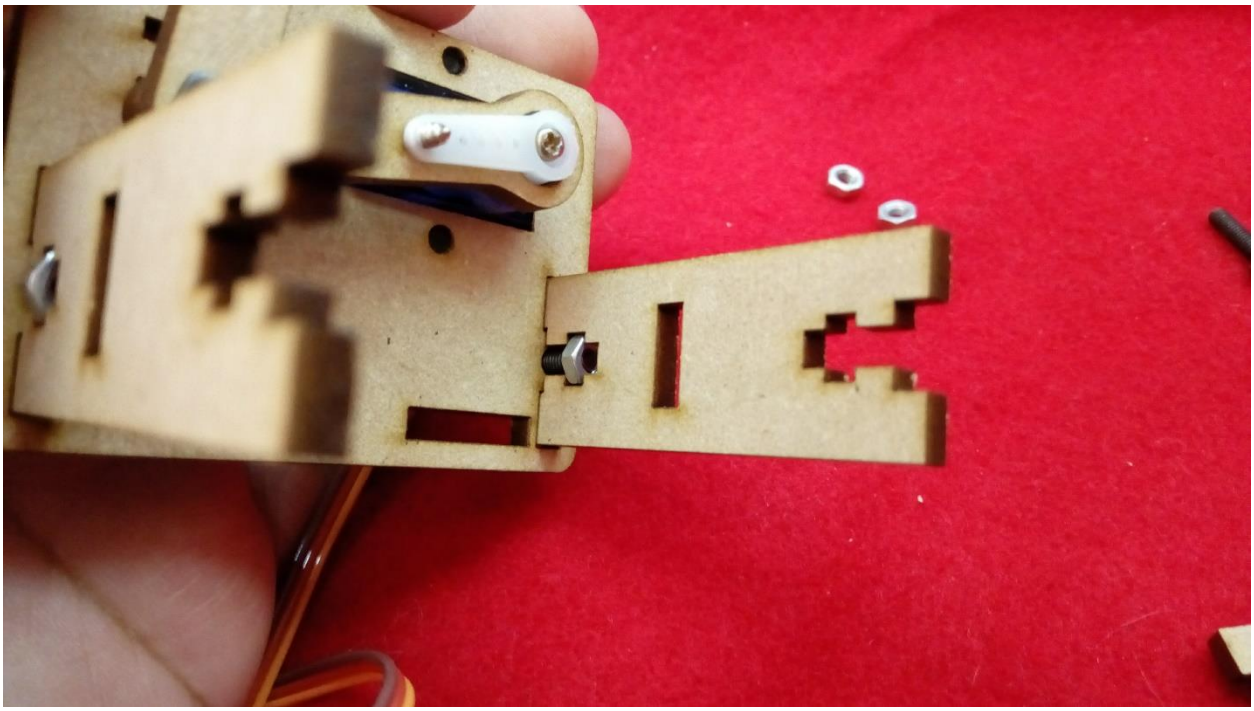


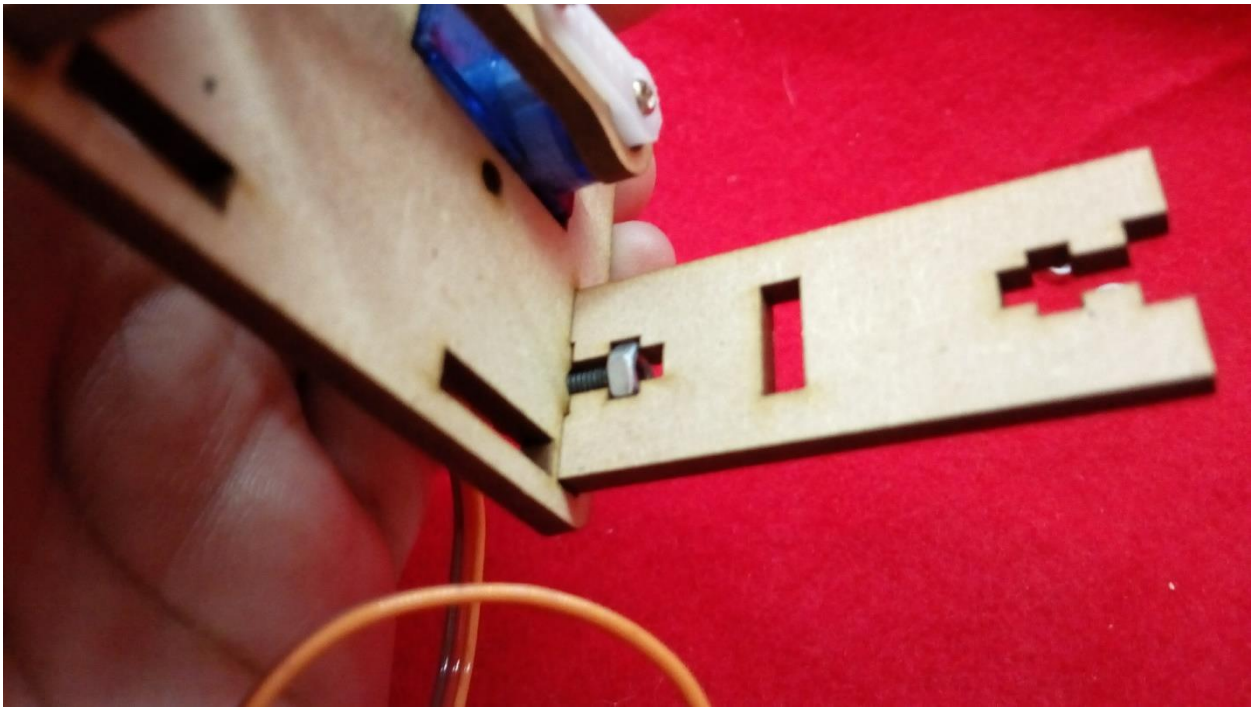


El siguiente paso consiste en unir todas las piezas, presente todas las partes que ha armado hasta este momento (ver imagen), tome las dos partes que tienen hecho un rectángulo pequeño e insértelo en el lado de la izquierda, asegurándose de que los tornillos de 12mm de la parte izquierda se empujan hacia la derecha insertándolos en la pieza y apriete los tornillos con una o dos vueltas, pero no todo.

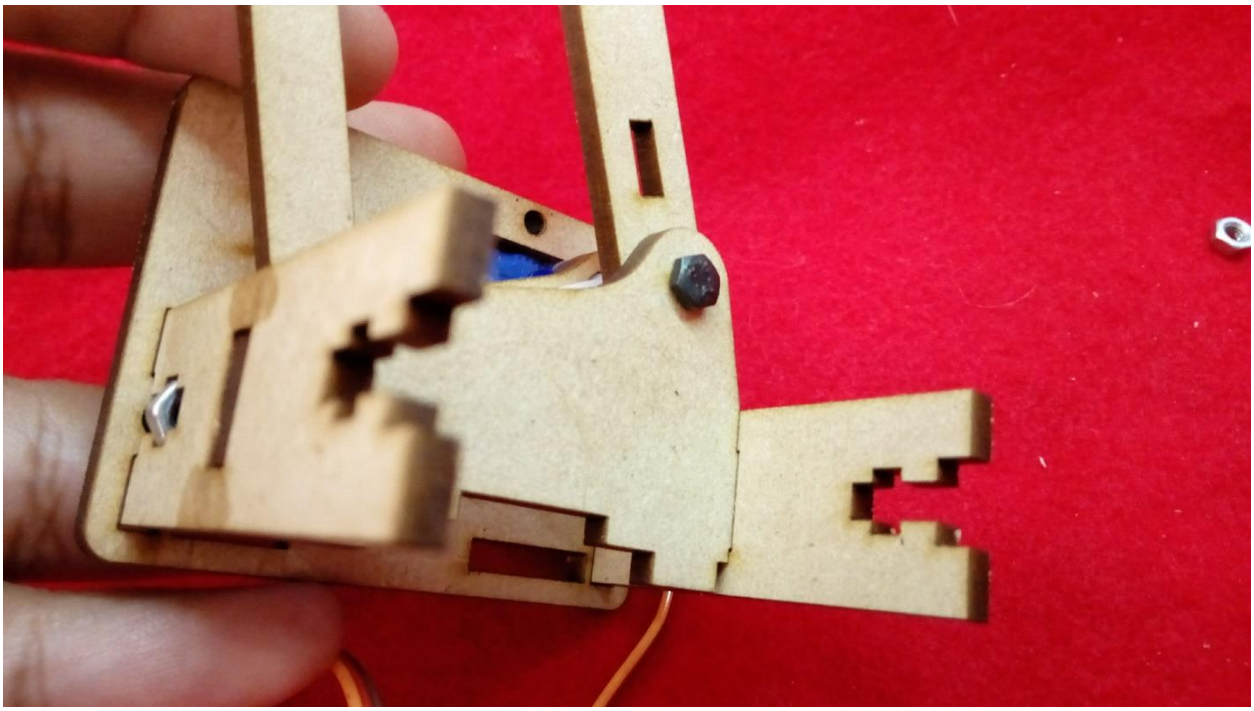


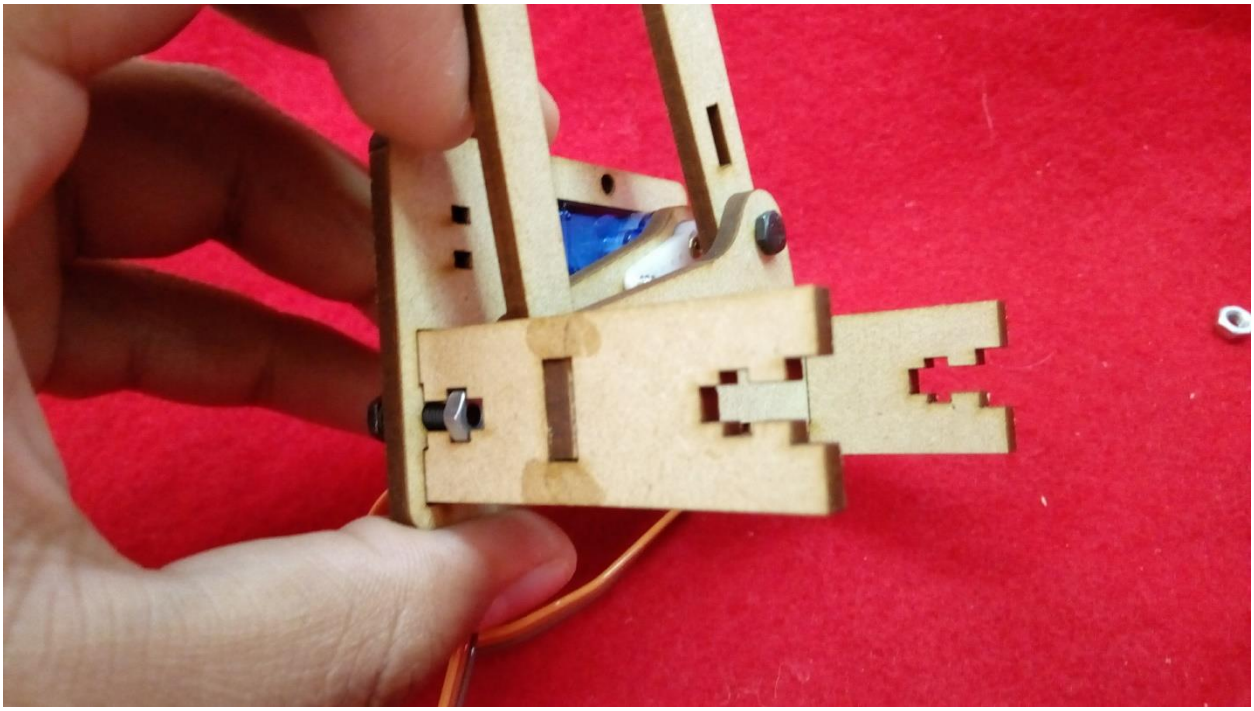


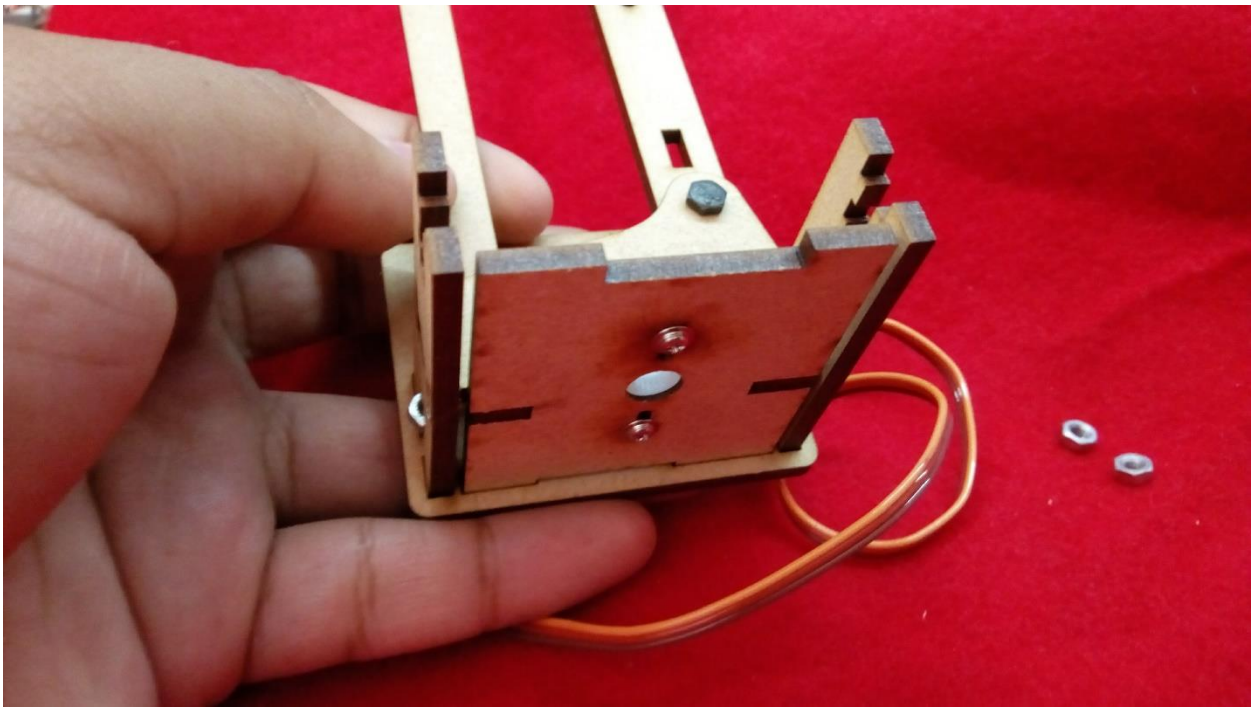




Ahora inserte con mucho cuidado la pieza “cerdo” en medio de las dos piezas que se colocaron. Una vez que se tienen juntas todas estas piezas puede insertar la parte de la base que se elaboró arriba, ahora si debe apretar los tornillos perfectamente.





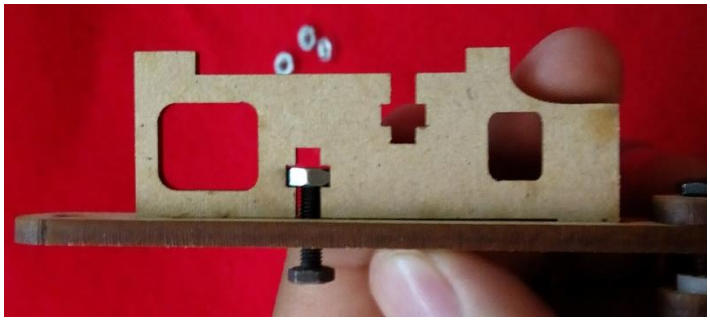


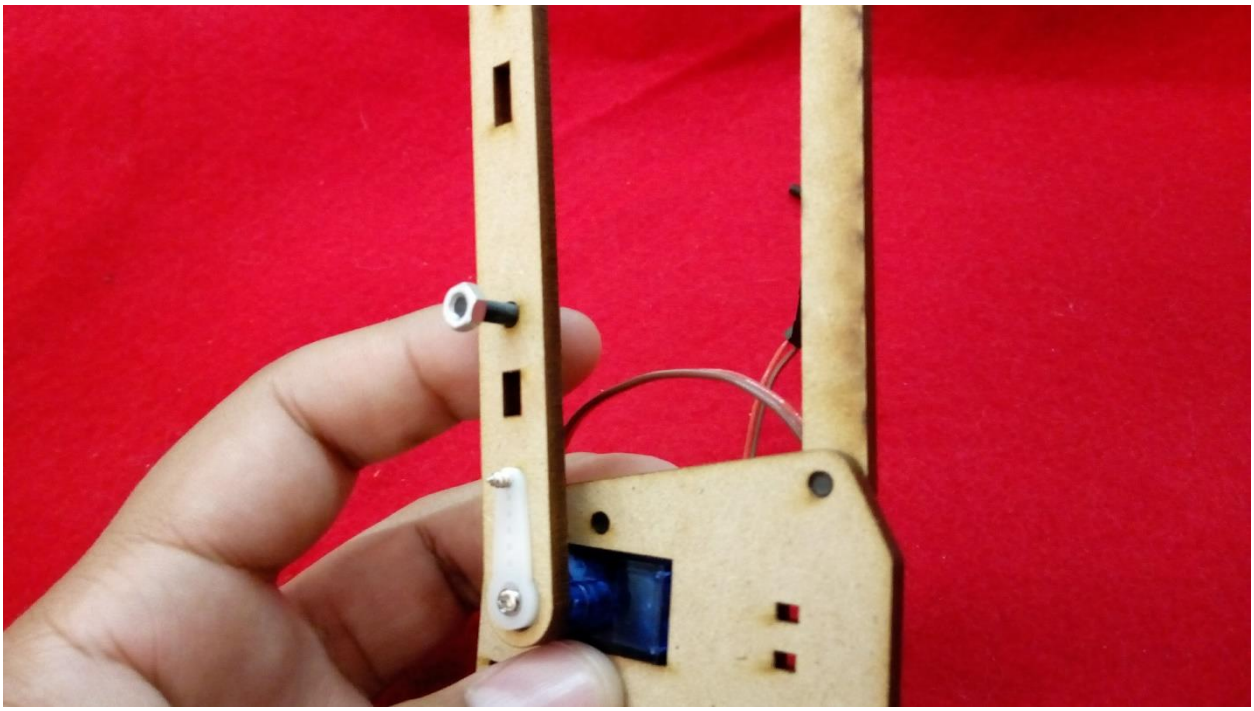
5.1.5 ARMADO DEL BRAZO – PARTE CENTRAL

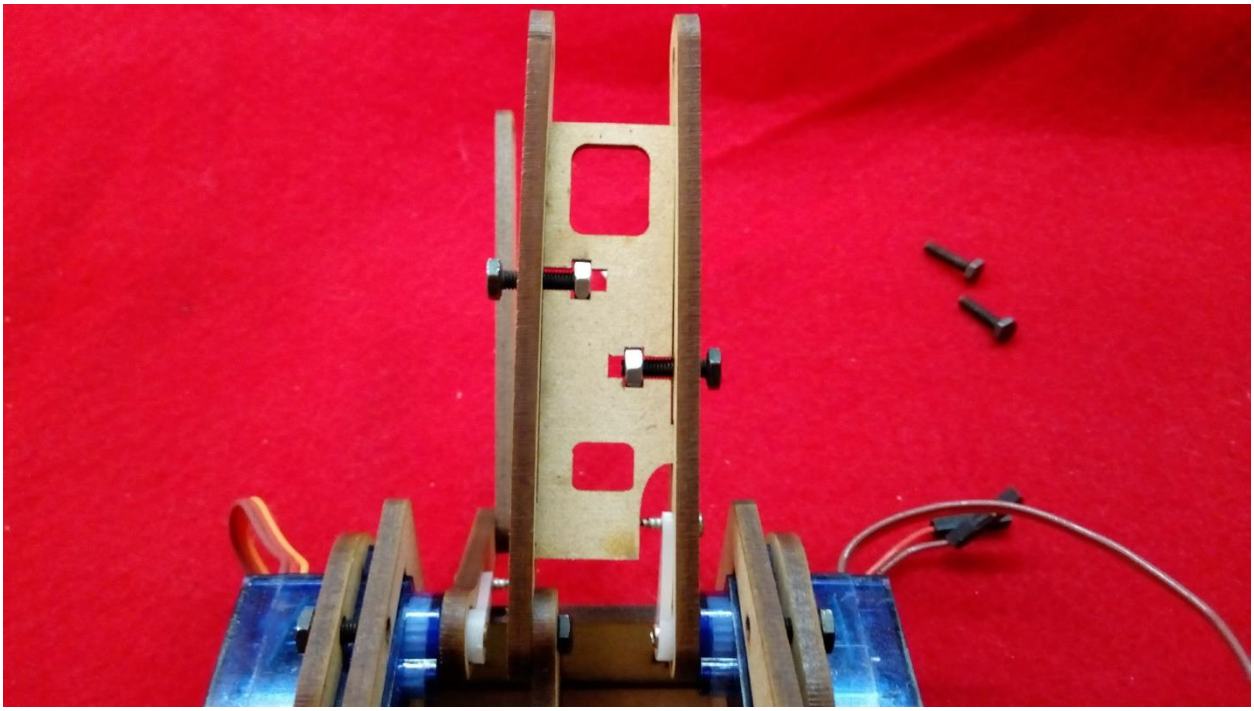
En esta parte se mostrará cómo armar la parte central del brazo, a continuación se lista el material necesario para la misma.

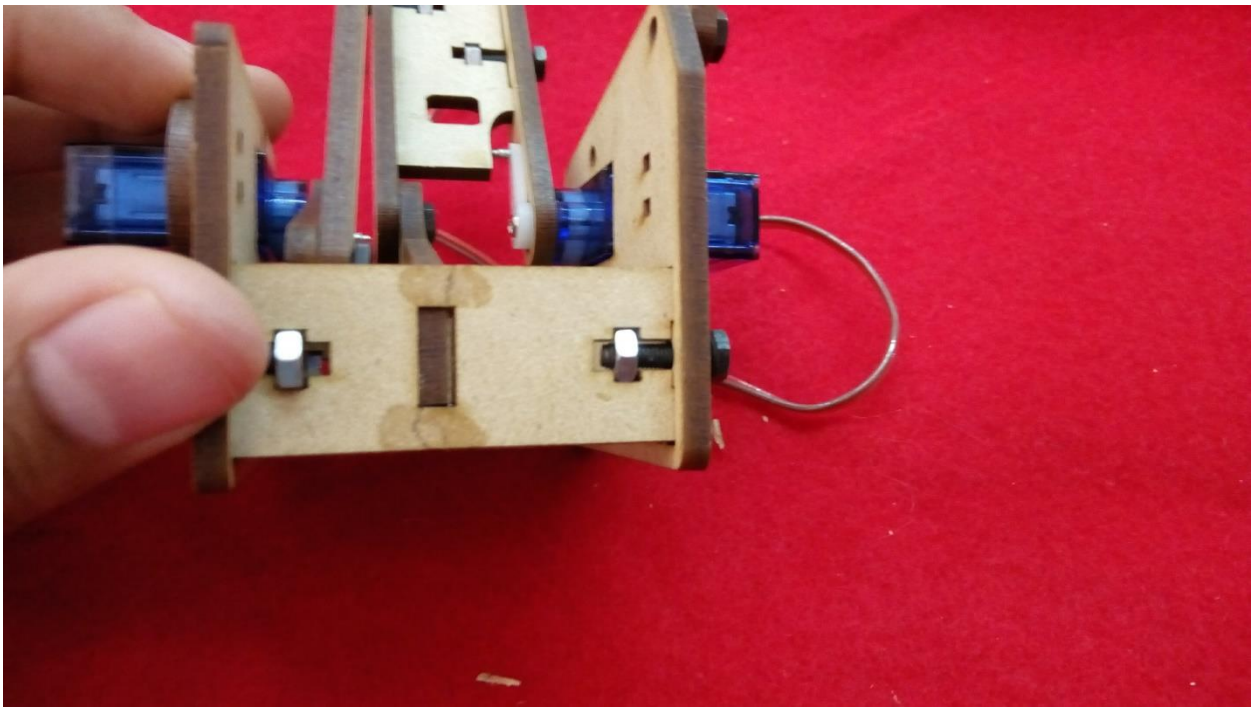
MATERIALES
2 Tornillos de 12mm
2 Tuercas

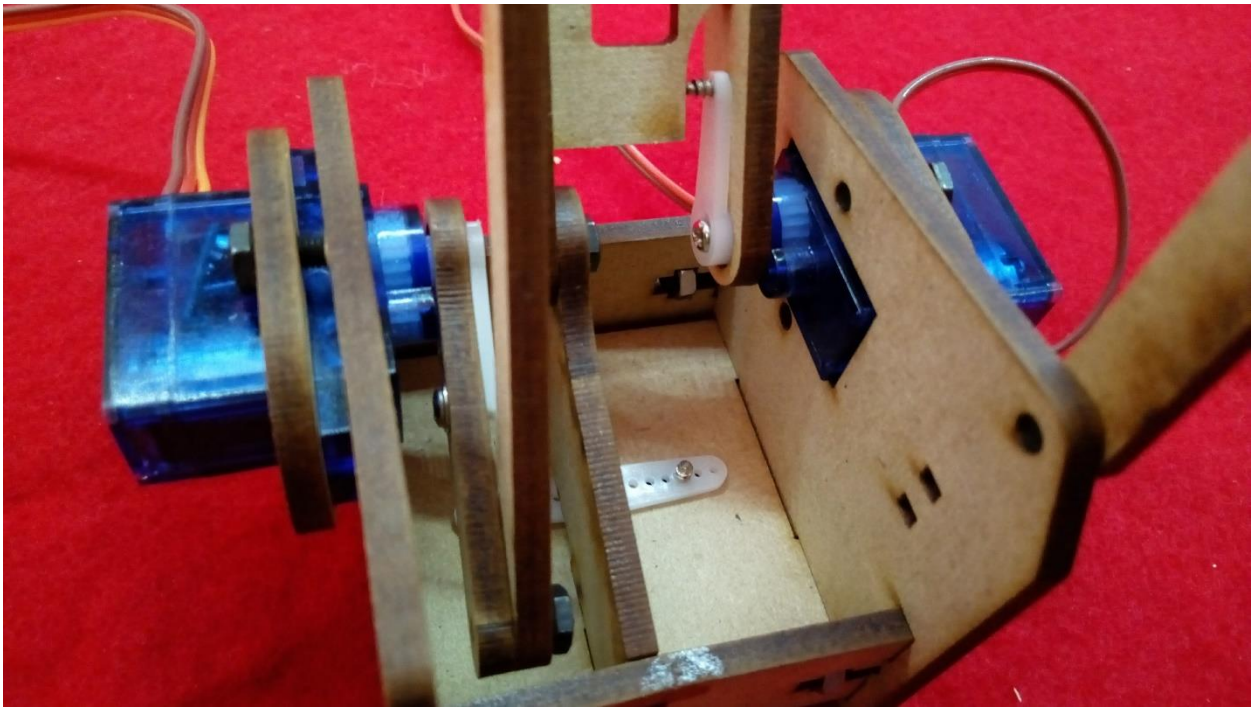
Inserte la parte central en la pieza que tiene dos pequeños agujeros cuadrados en la parte de la palanca izquierda, inserte un tronillo de 12mm con su respectiva tuerca, apriete el tronillo, luego inserte la parte derecha con el otro tornillo y tuerca, ahora encaje todas las piezas como se muestra en las imágenes.

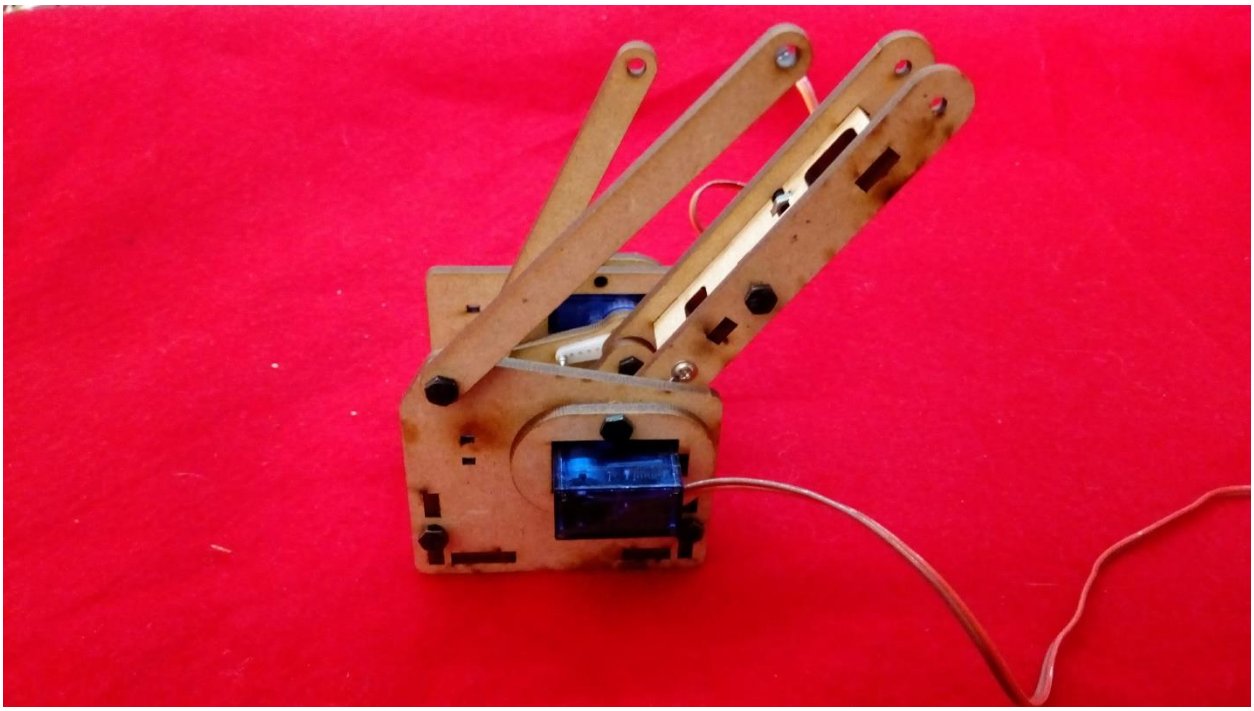




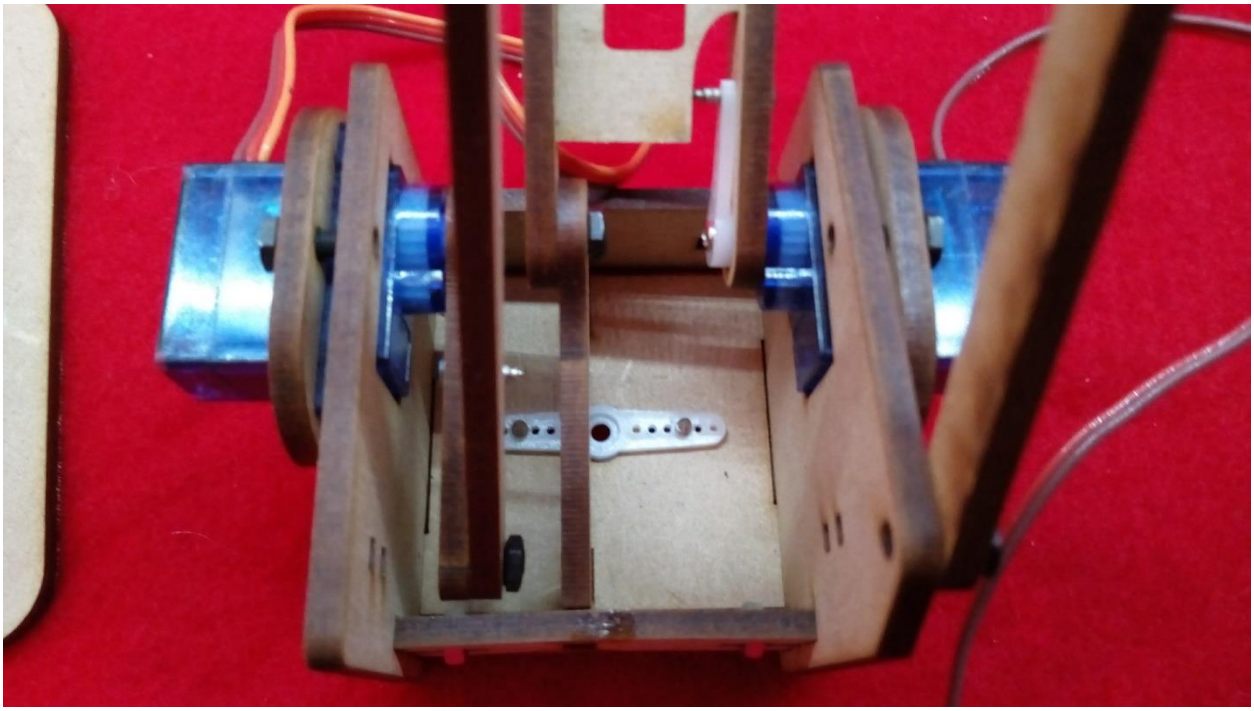


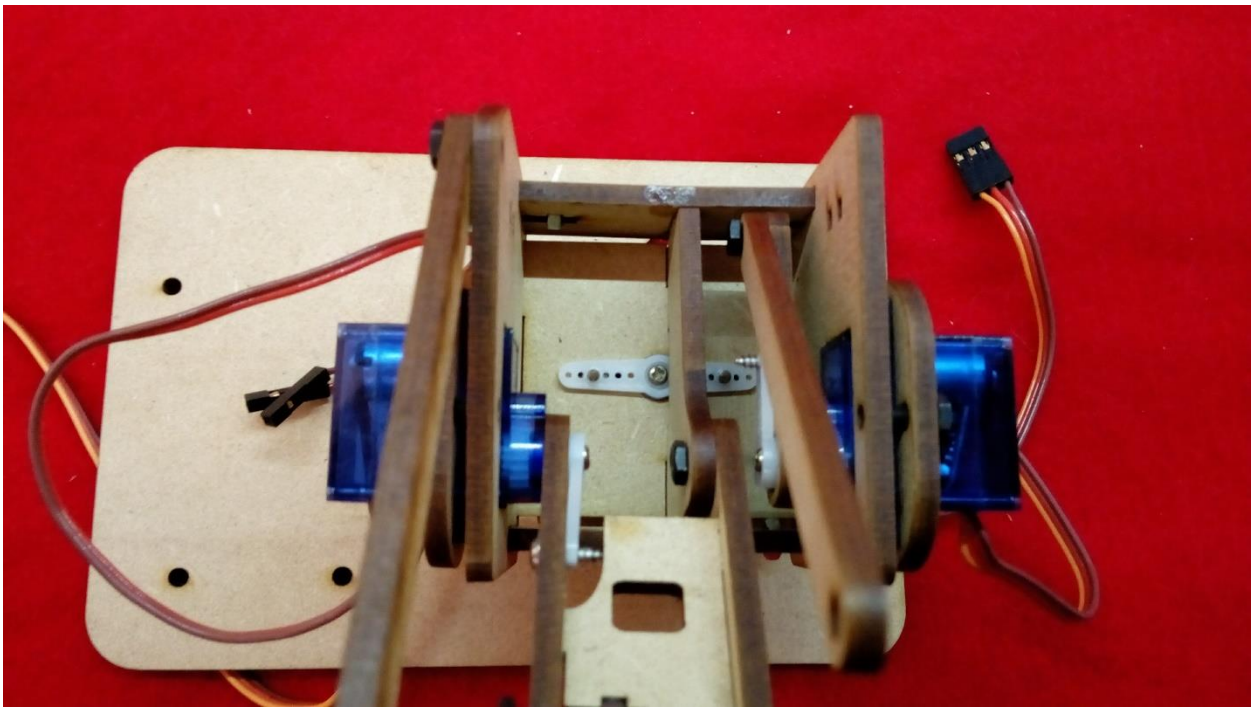


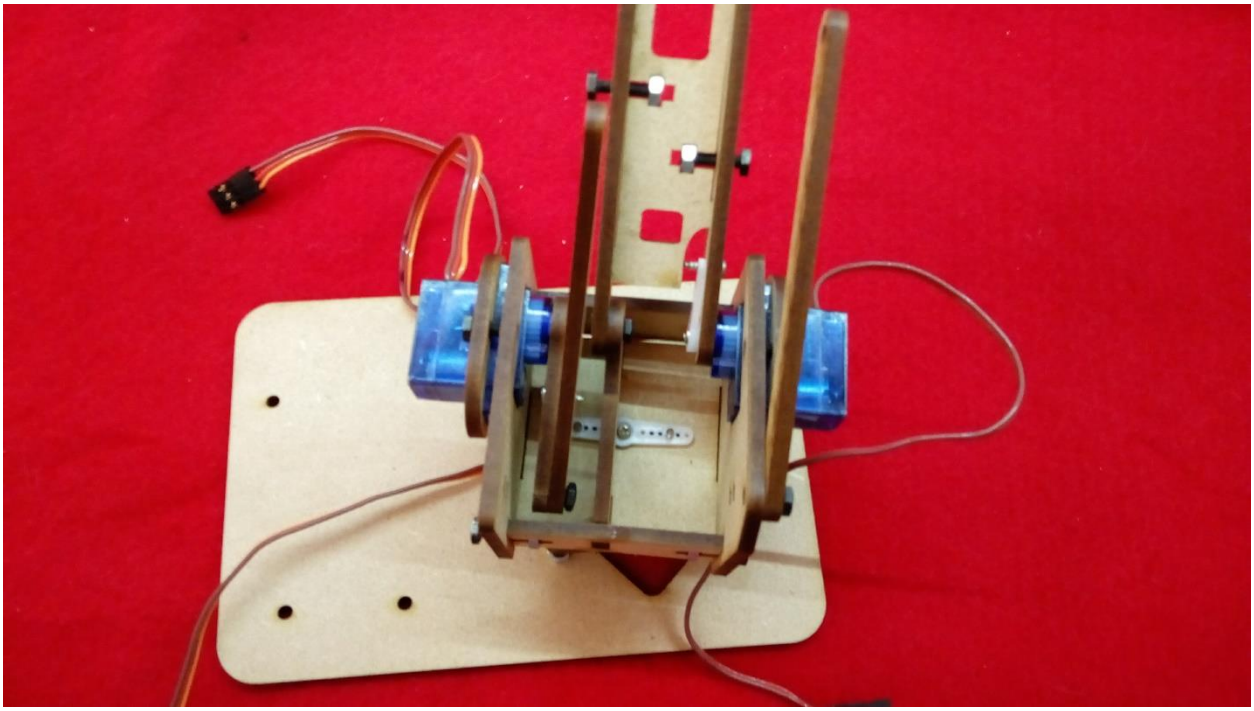




Ahora hay que insertar la pieza terminada anteriormente en el servomotor de la base, recuerde que el servomotor de la base tiene que estar a 90 grados, inserte esta pieza centralmente ya que debe girar de 0 a 180 grados, sujétela con el tornillo pequeño del servomotor.



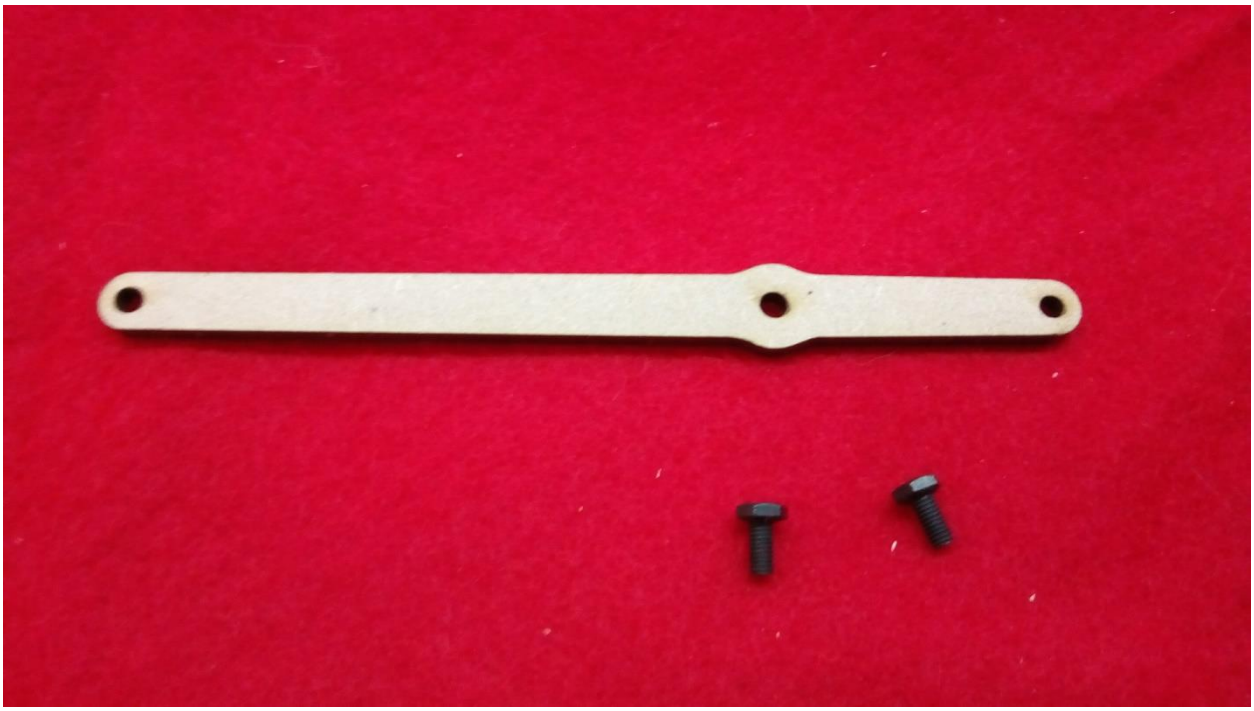




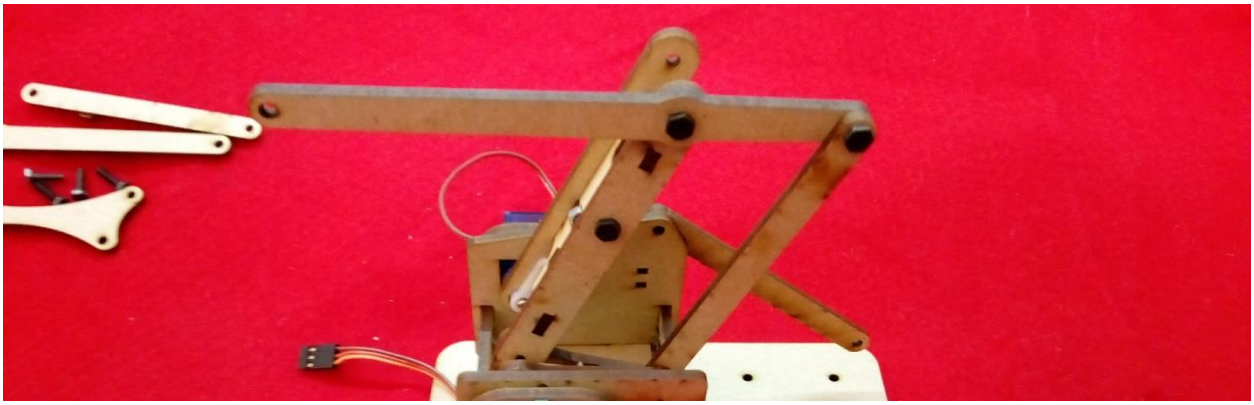
5.1.6 ARMADO DEL BRAZO – ANTEBRAZO IZQUIERDO Y DERECHO

En esta parte se mostrará cómo armar el antebrazo izquierdo y derecho del brazo, a continuación se lista el material necesario para la misma.

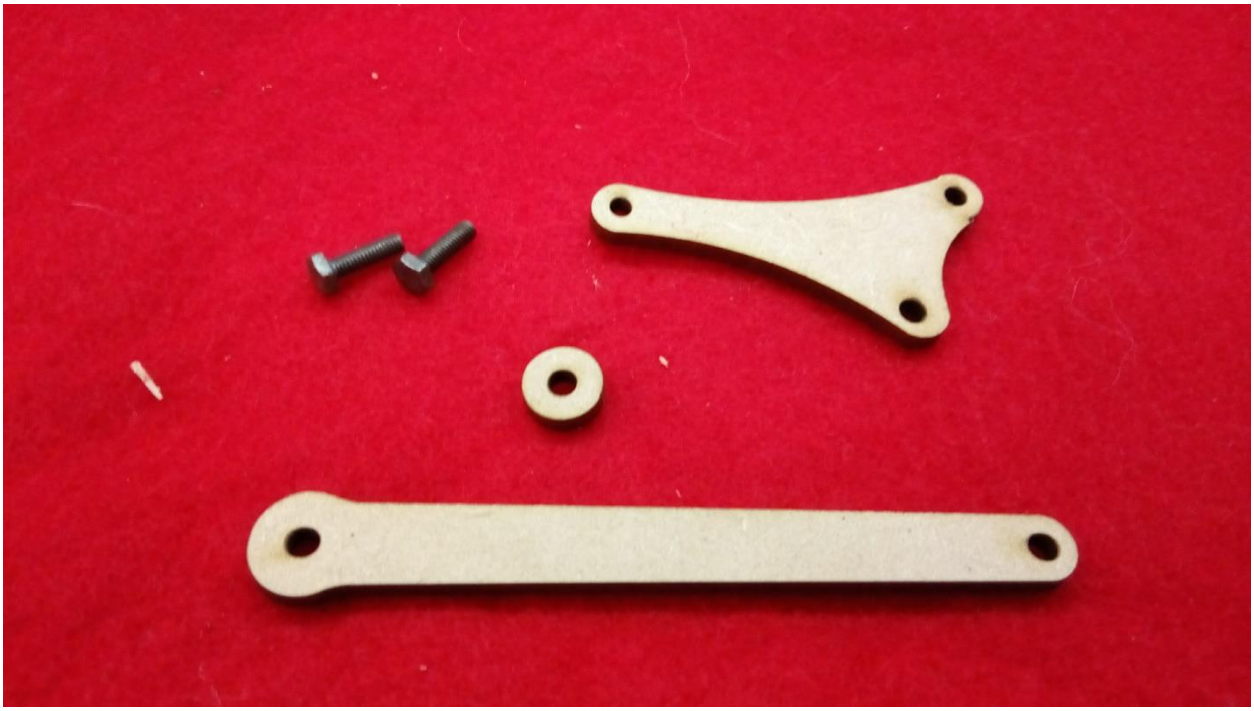
MATERIALES
3 Tornillos de 6mm
2 Tornillos de 10mm
1 Espaciador (círculo pequeño con agujero)

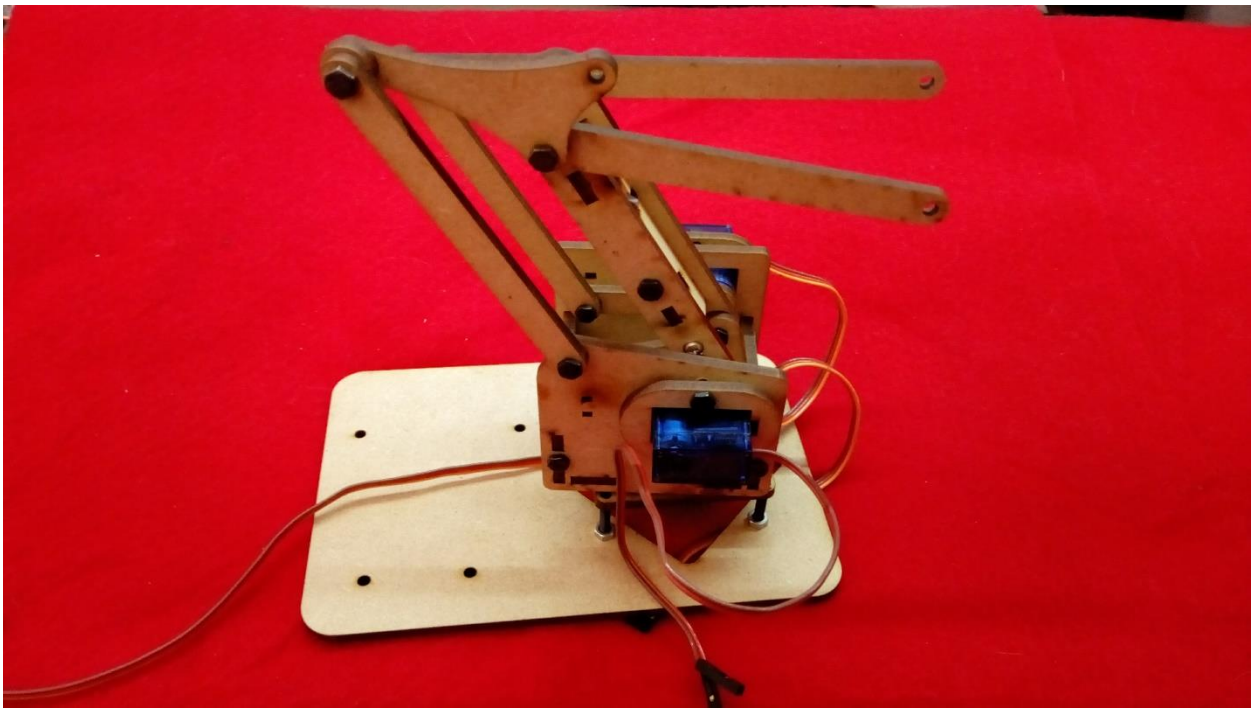


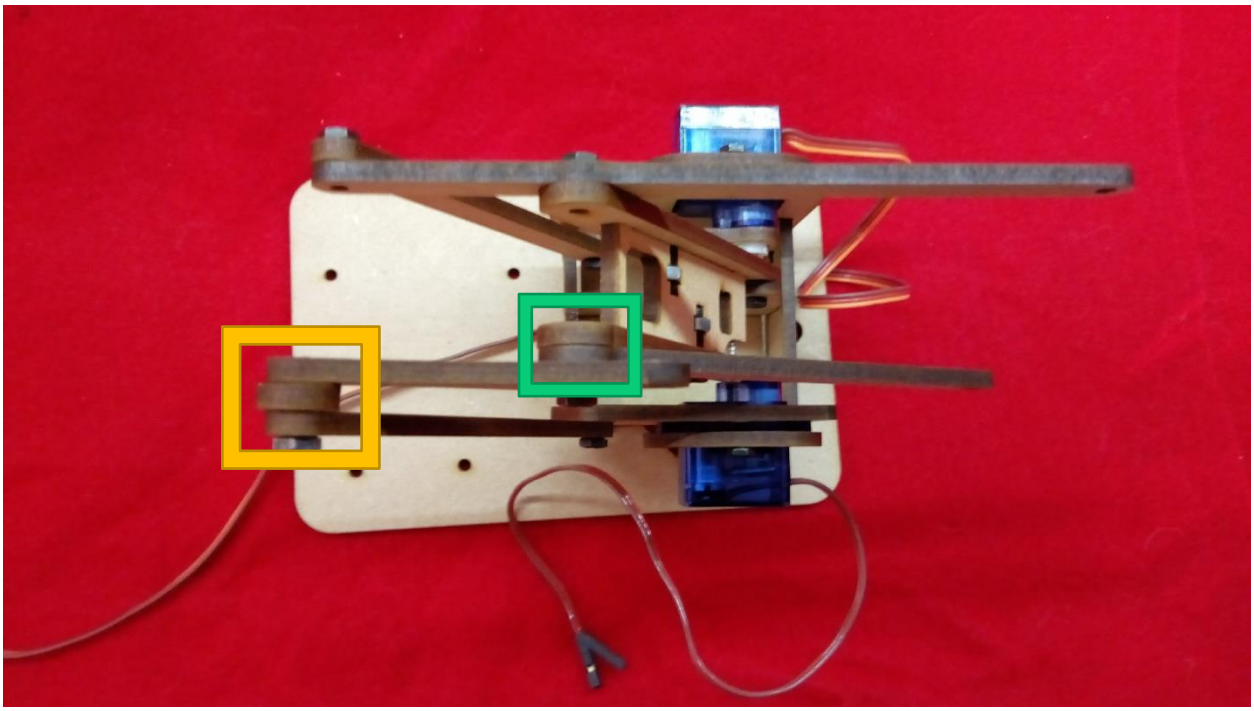
Insertar la pieza larga entre las dos palancas del lado izquierdo y sujetar con tronillos de 6mm.



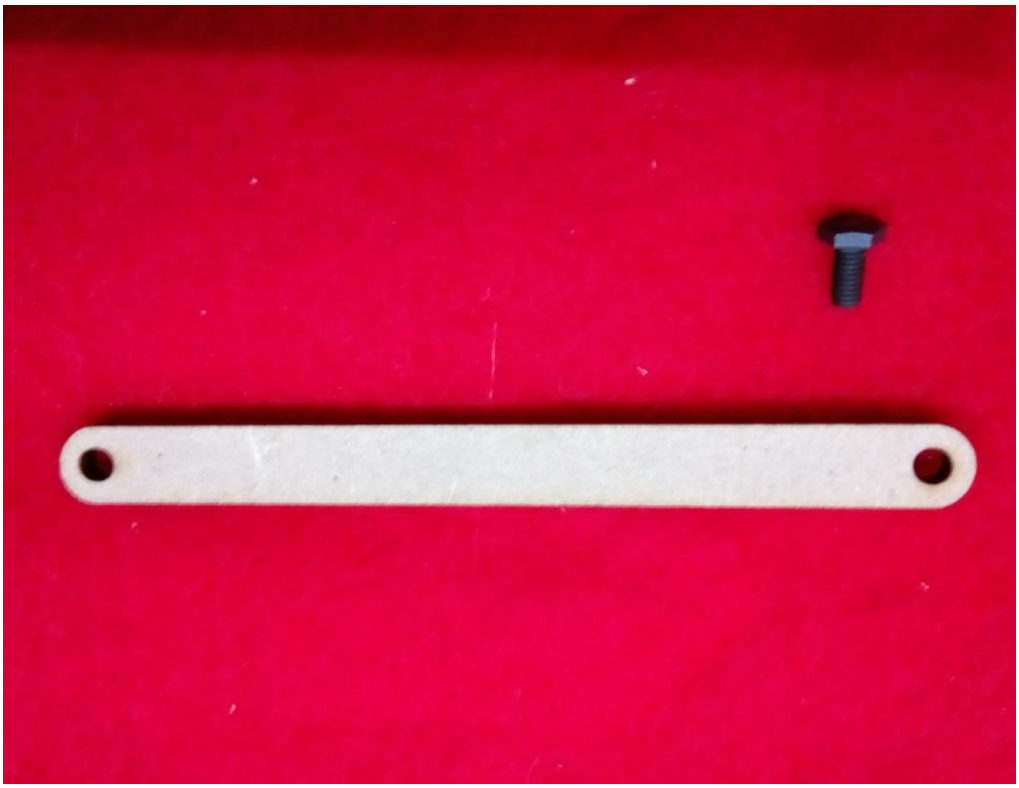
En el antebrazo derecho se requiere un espaciador y utilizar los dos tronillos de 10mm para sujetar las piezas, es una pieza en forma de “Y” y otra alargada semi-redonda de una punta.

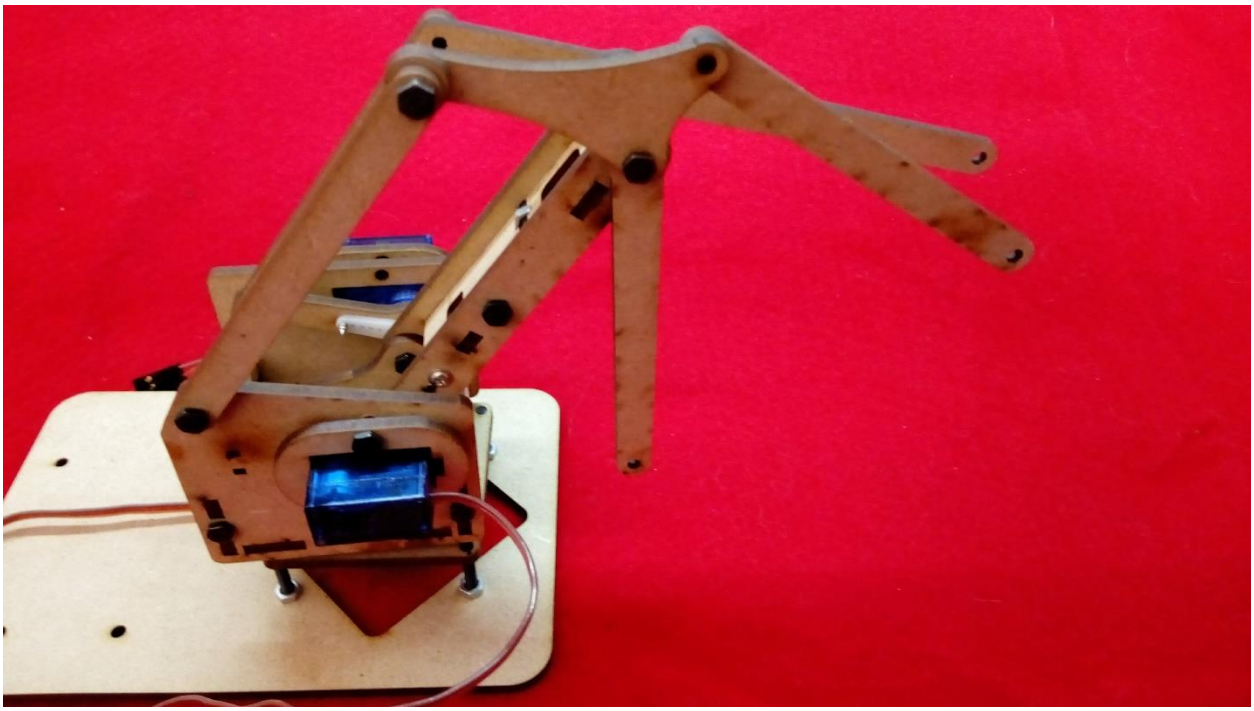






En medio de la palanca derecha y la pieza en forma de “Y” se coloca el espaciador (marco amarillo), del otro lado (marco verde) la pieza alargada se coloca en medio de la pieza “Y” y la palanca central. Al final solo se debe sujetar la pieza alargada a la pieza “Y” con un tronillo de 6mm, la cabeza del tronillo va por dentro.



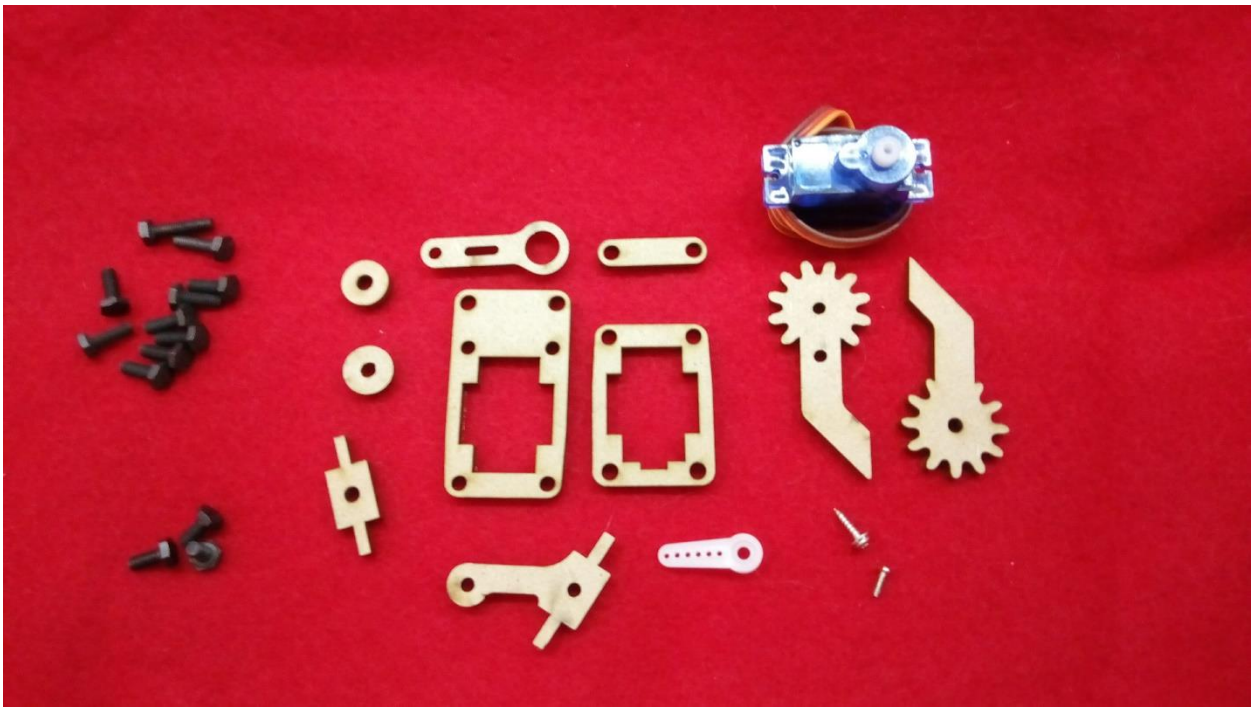


5.1.7 ARMADO DEL BRAZO – MANO

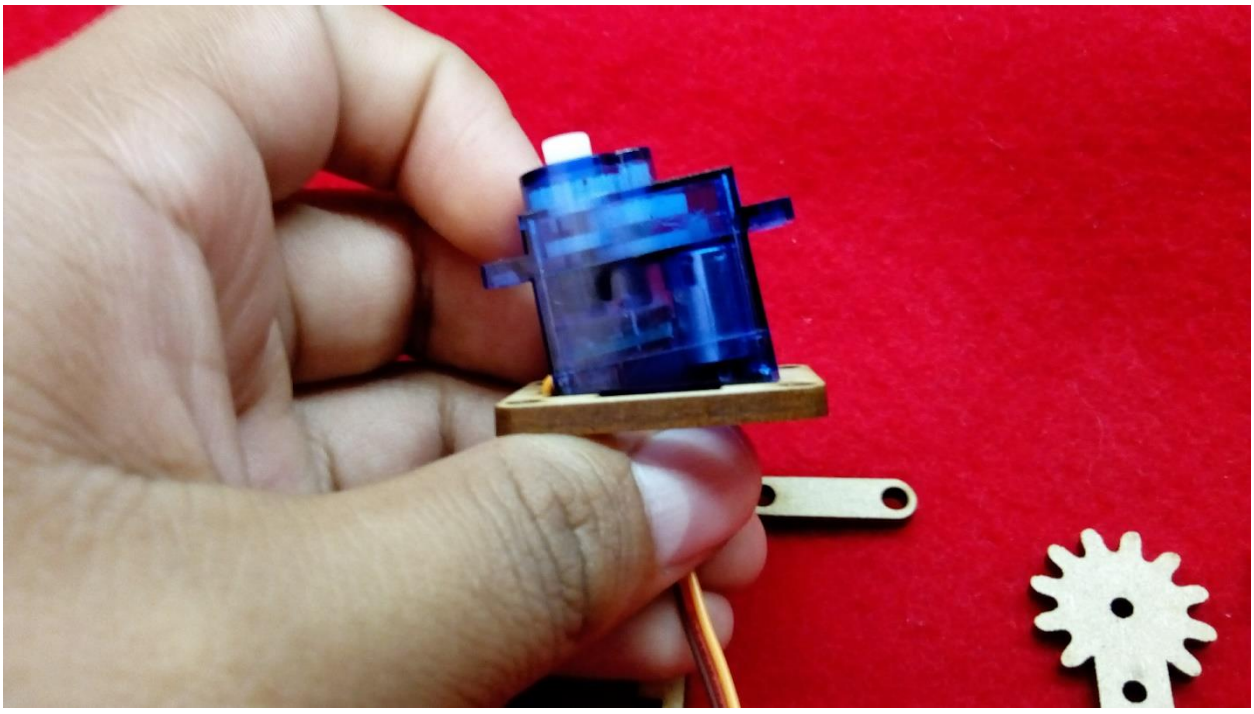
En esta parte se mostrará cómo armar la mano del brazo, a continuación se lista el material necesario para la misma.

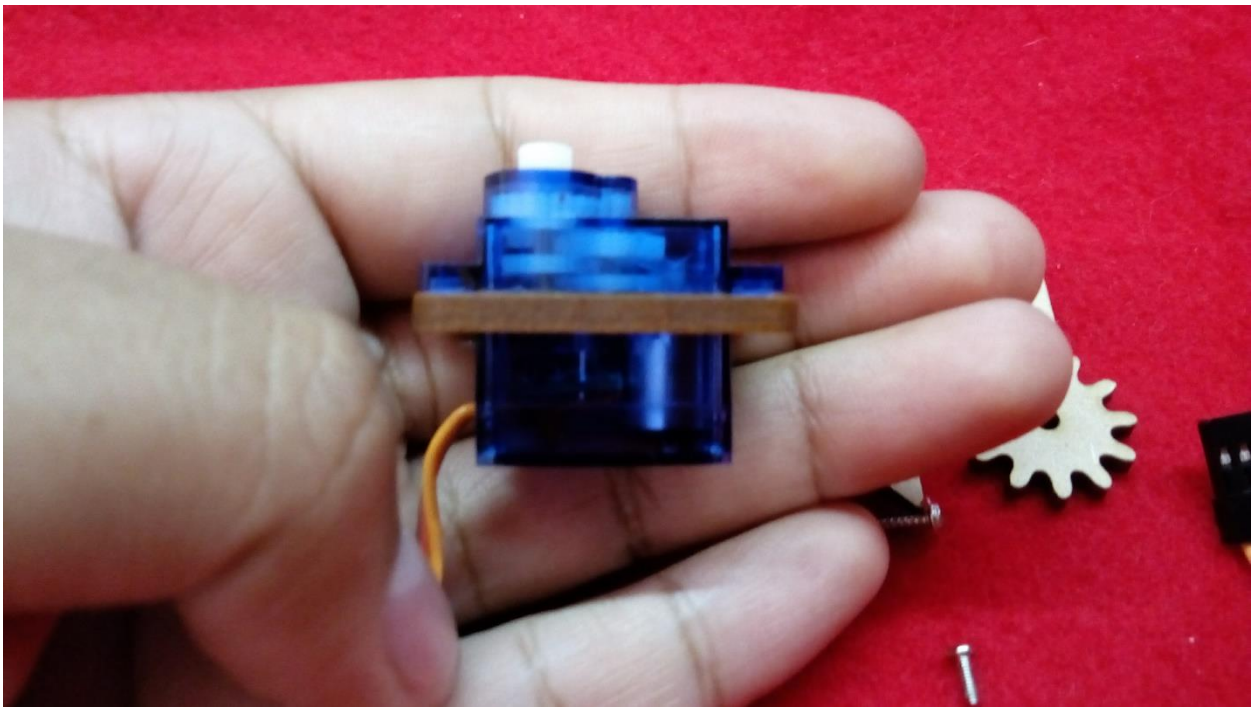
MATERIALES
8 Tornillos de 8mm
3 Tornillos de 6mm
1 Tornillo de 12mm
1 Tornillo de 10mm
1 Servomotor
Tornillo largo y corto del servomotor
Piezas de la mano (ver imagen)

El servomotor debe de estar a 90 grados, las imágenes se muestran abajo.

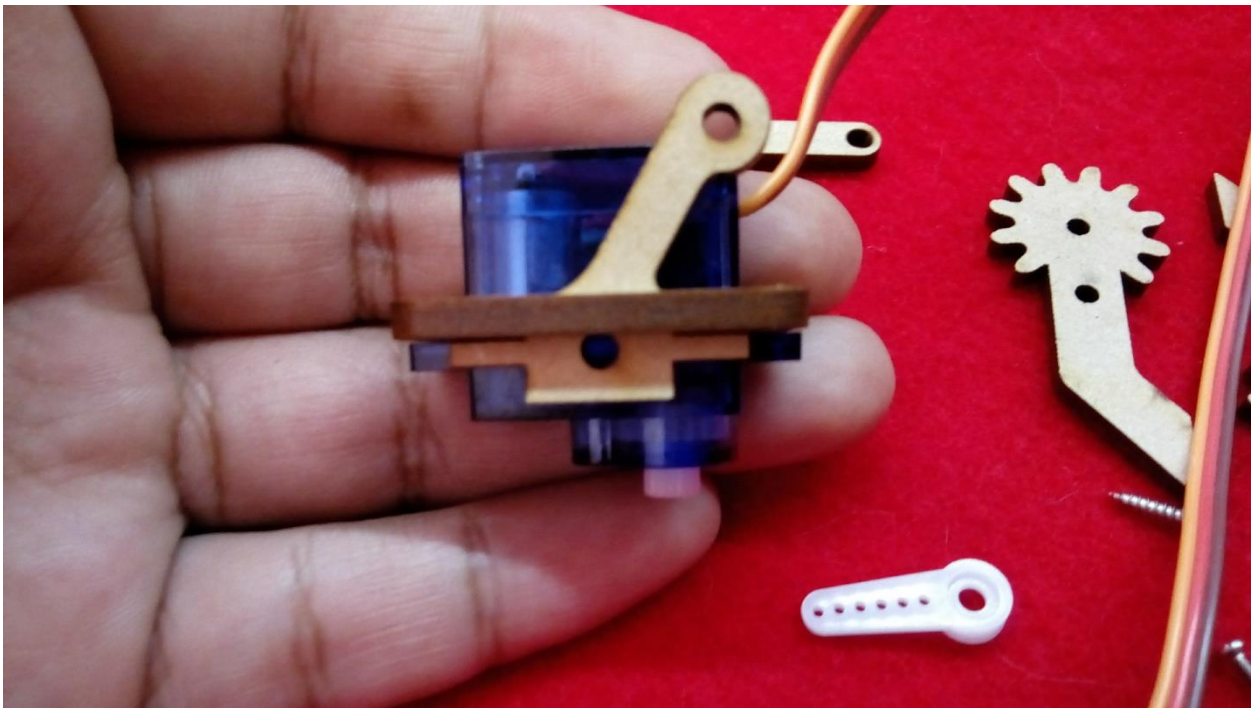


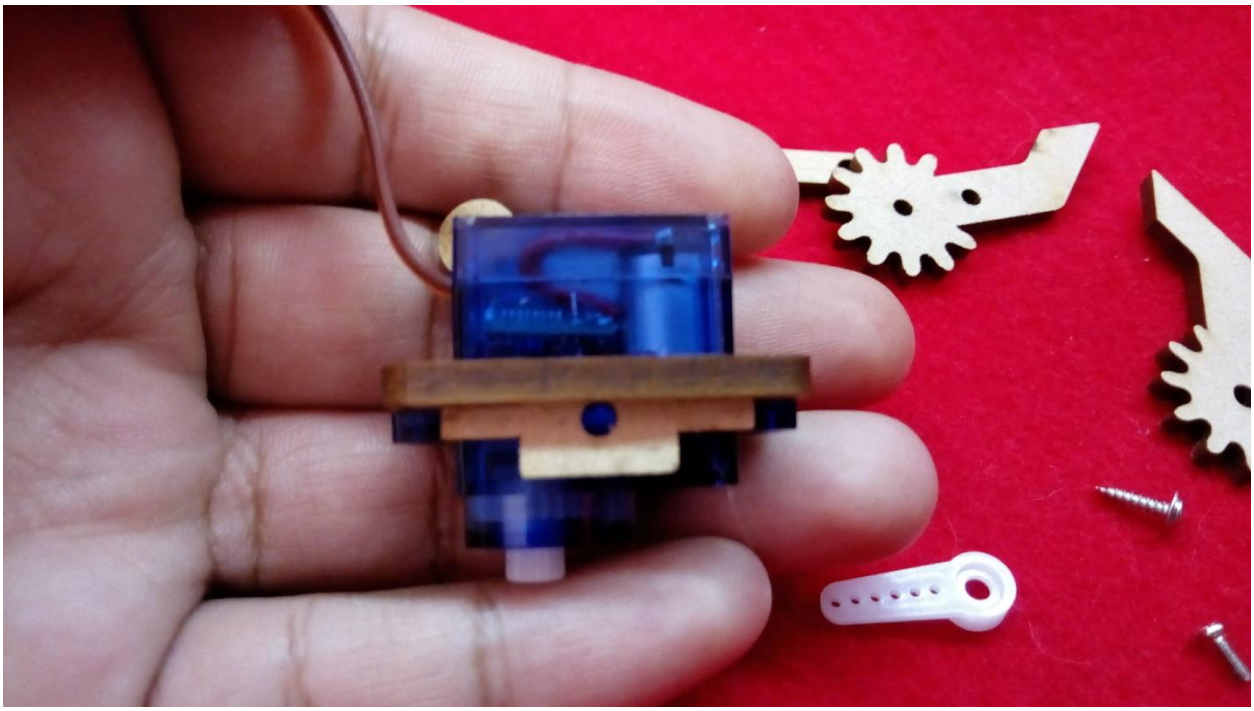
De las dos piezas rectangulares, utilice la pieza más corta, este será el collar que se le pondrá al servomotor.



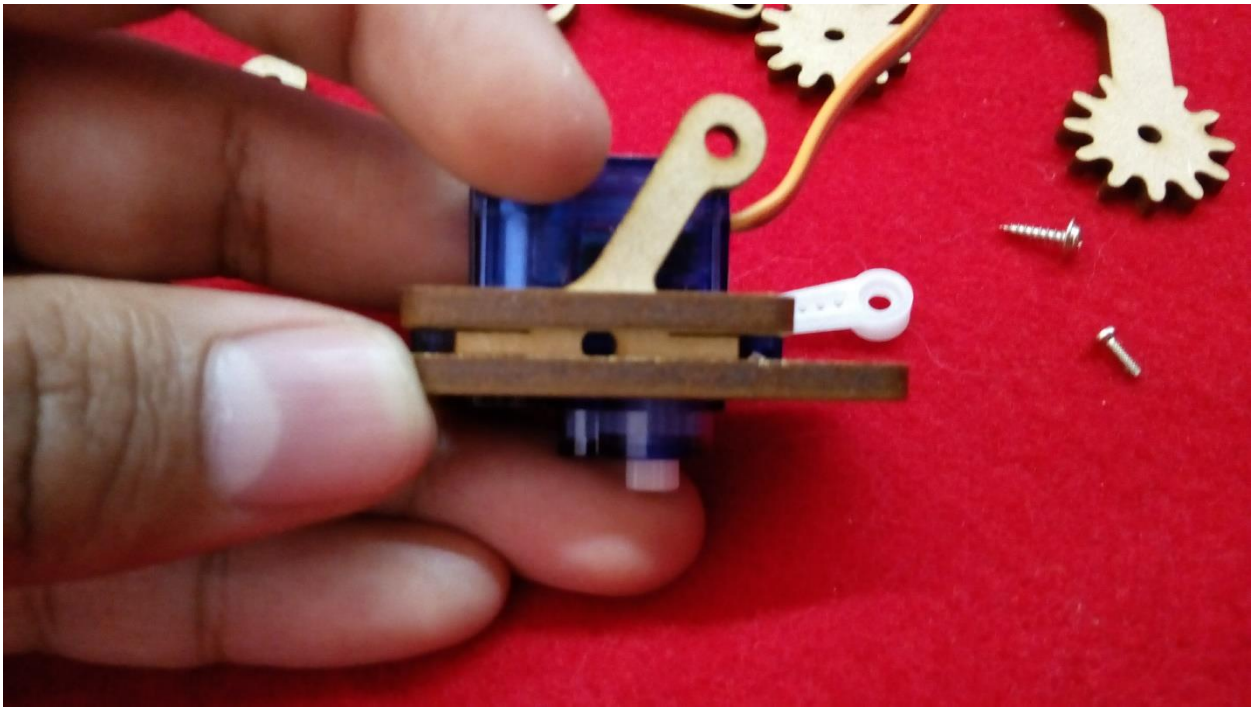


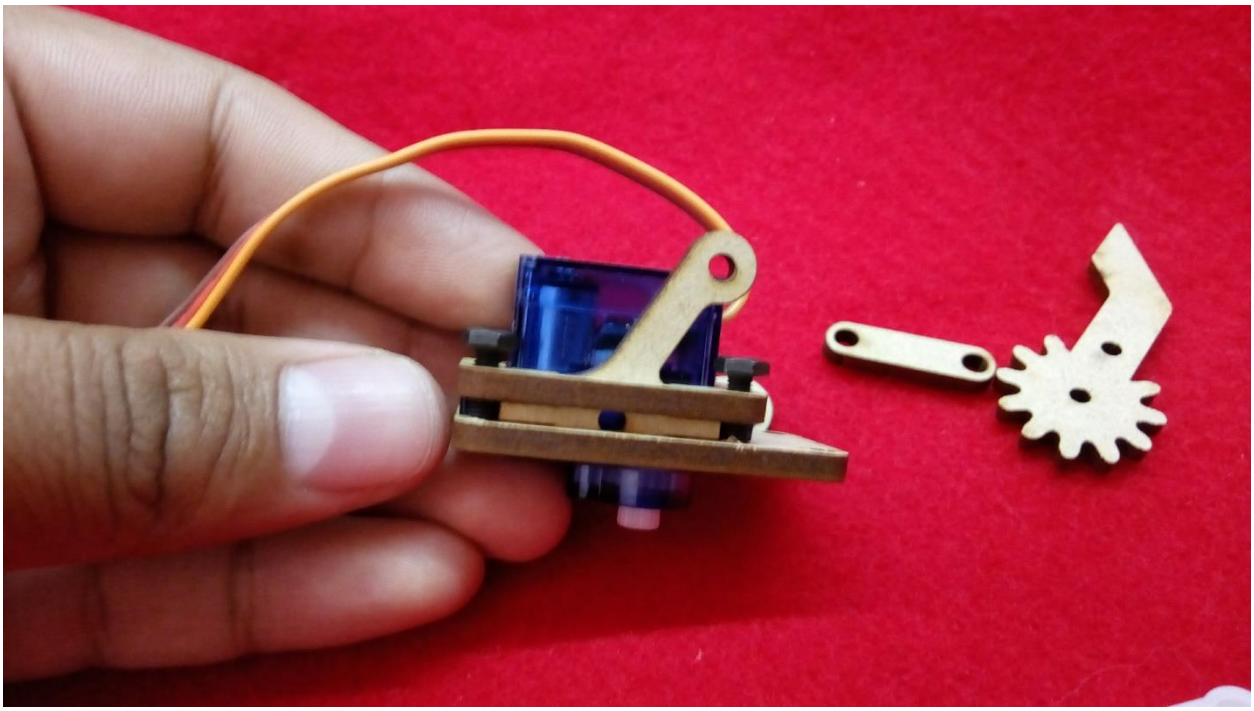
A continuación utilice las piezas delgadas e insértelas en el collar del servomotor, guíese por las siguientes imágenes.

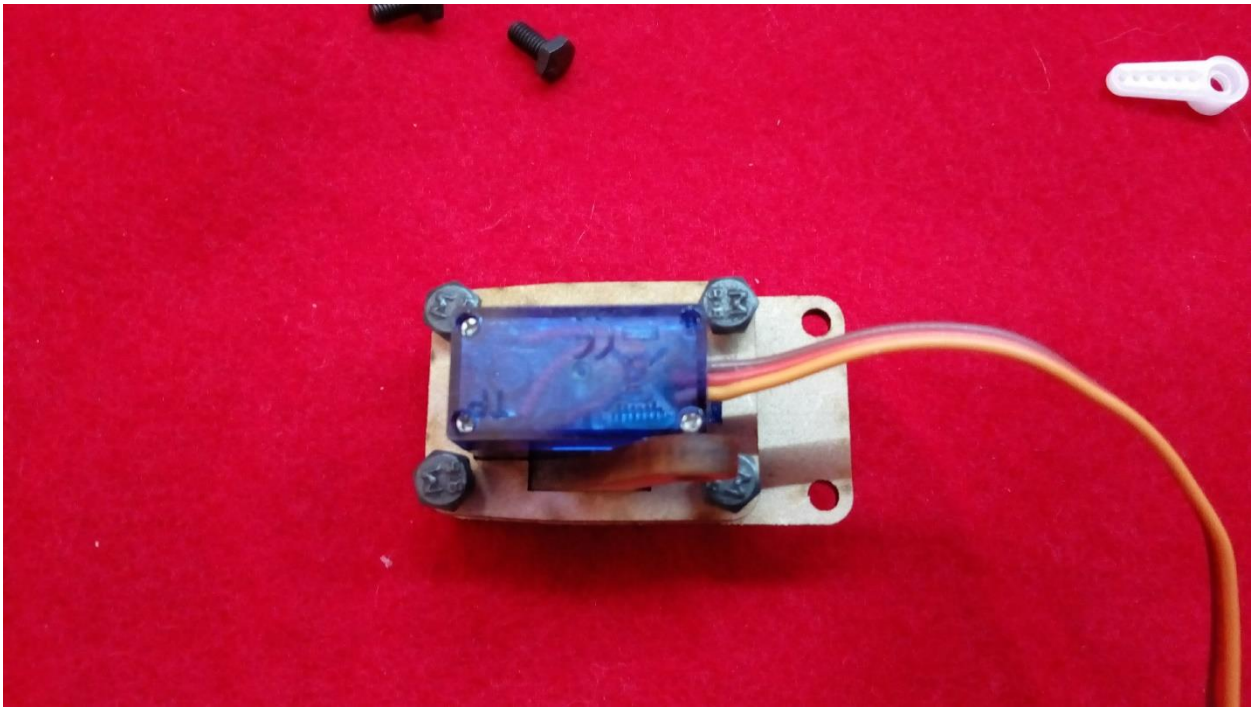




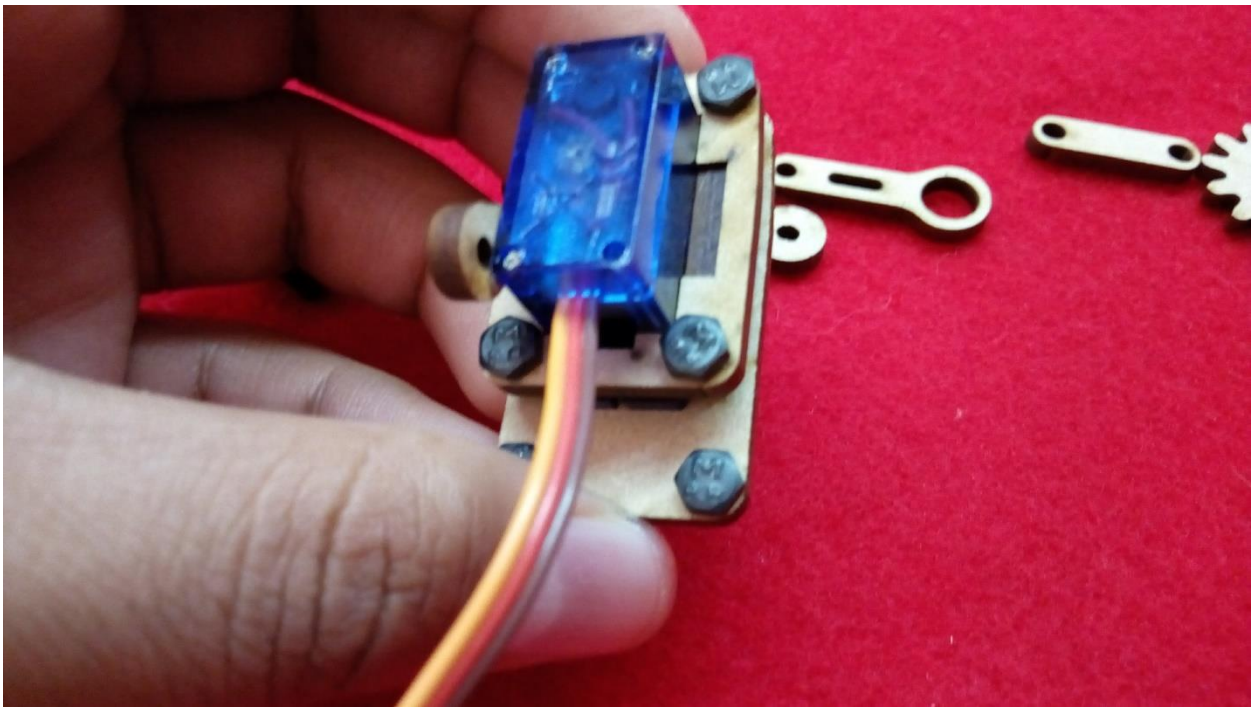
Ahora la parte rectangular más grande se debe de colocar sobre la parte inferior de la pieza que acaba de hacer, sujete la pieza con los tornillos de 8mm, apriete pero no demasiado.

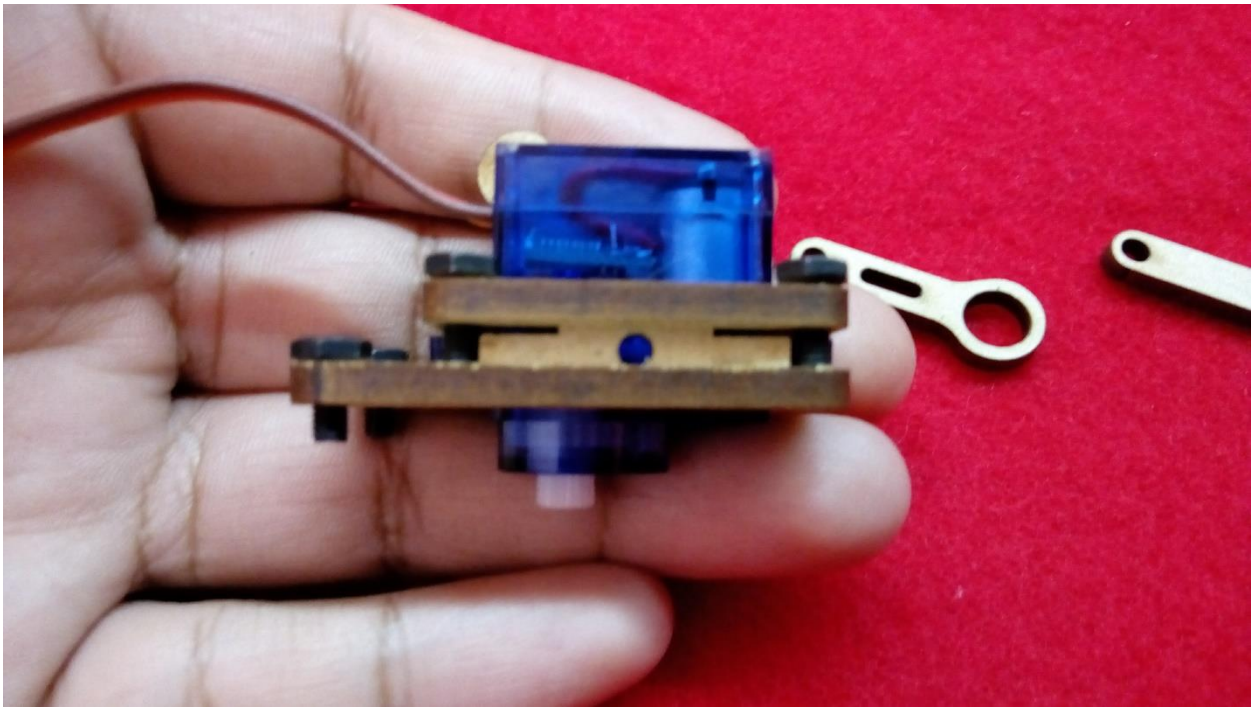


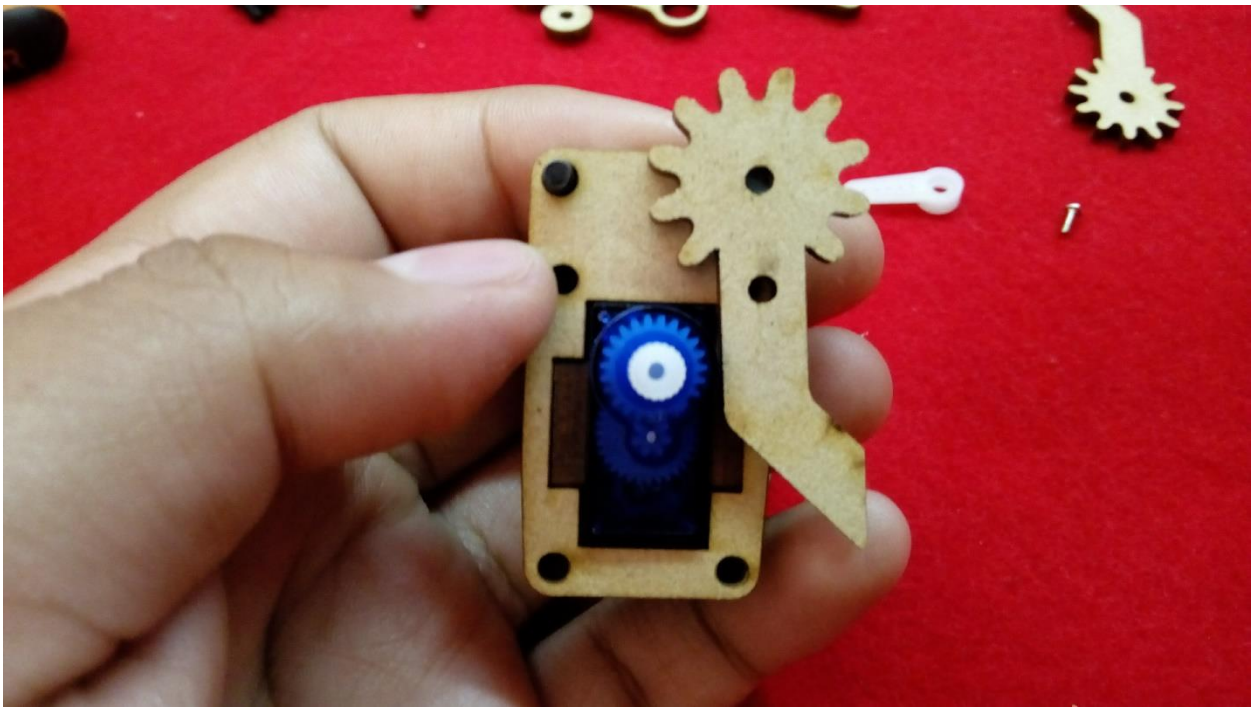


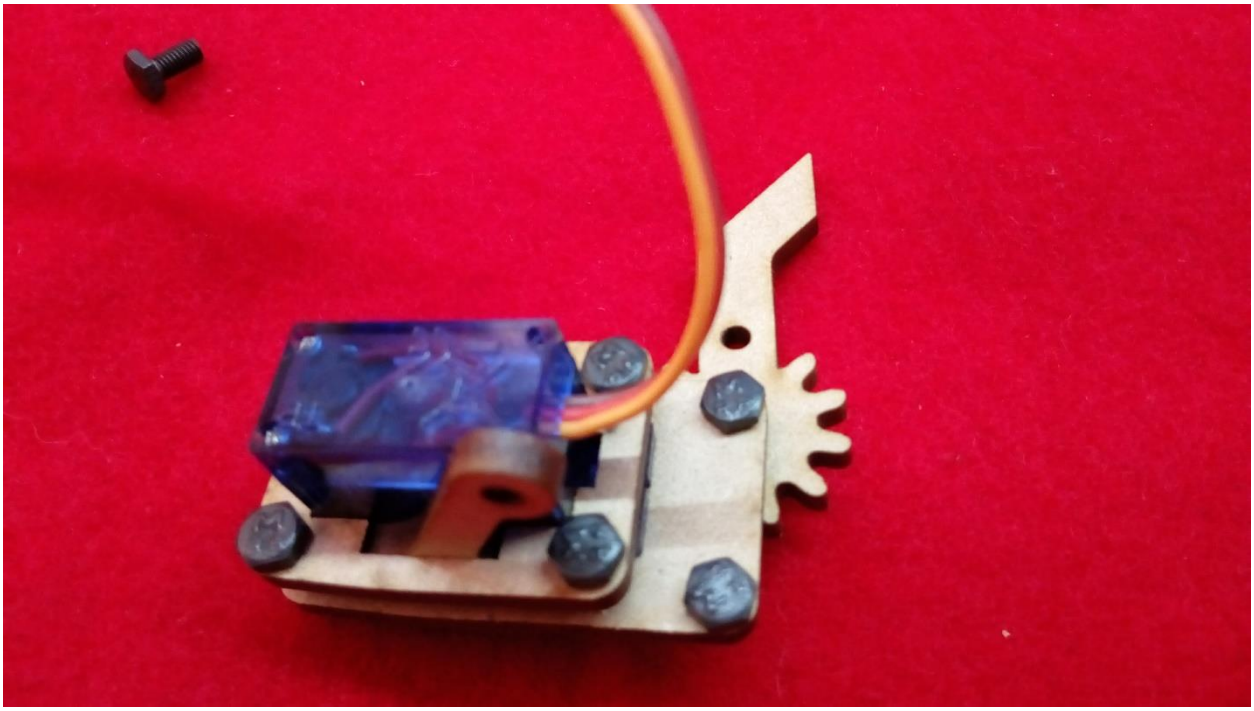


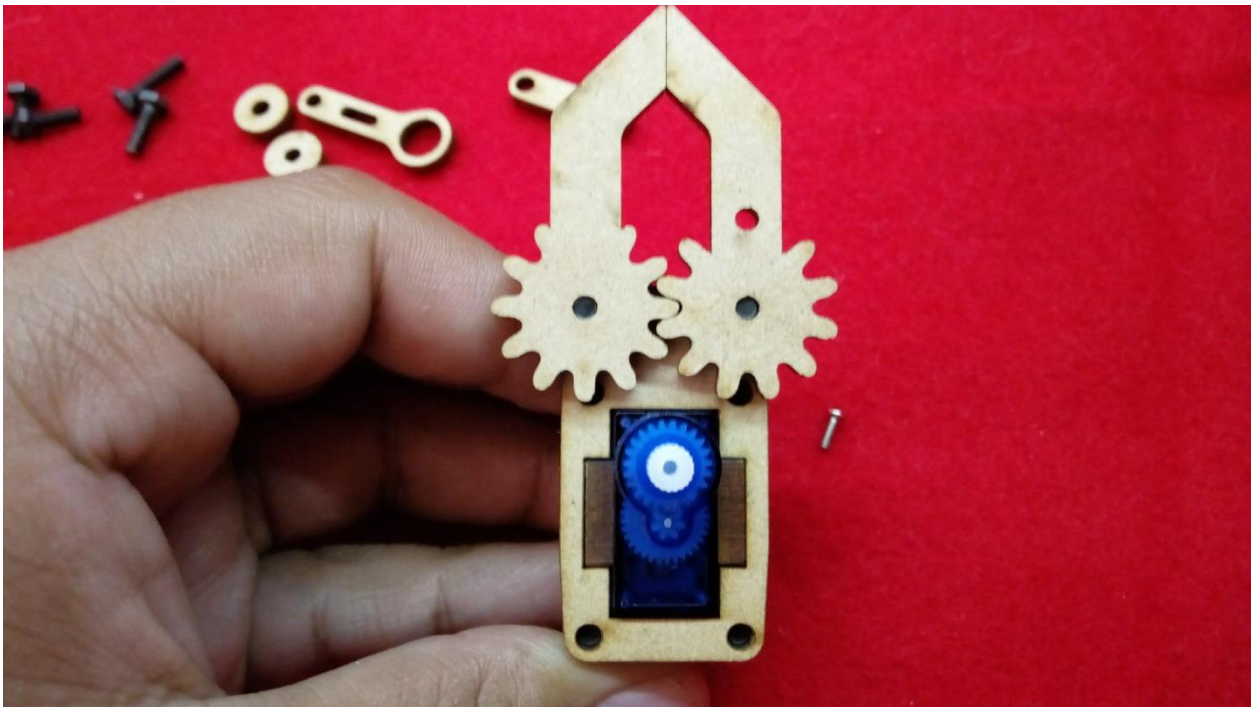
Fije las dos pinzas con la pieza anterior, para ello tome los dos tornillos de 6mm, no apriete demasiado, recuerde que tiene que ser móvil.



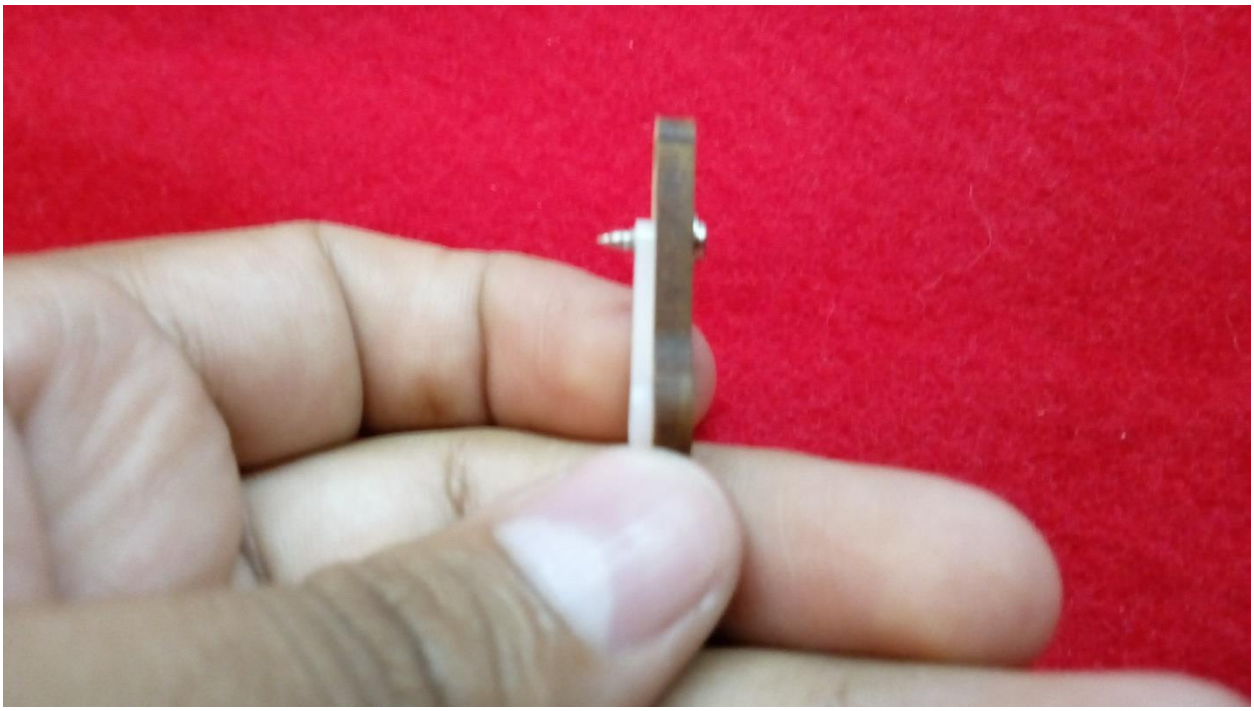




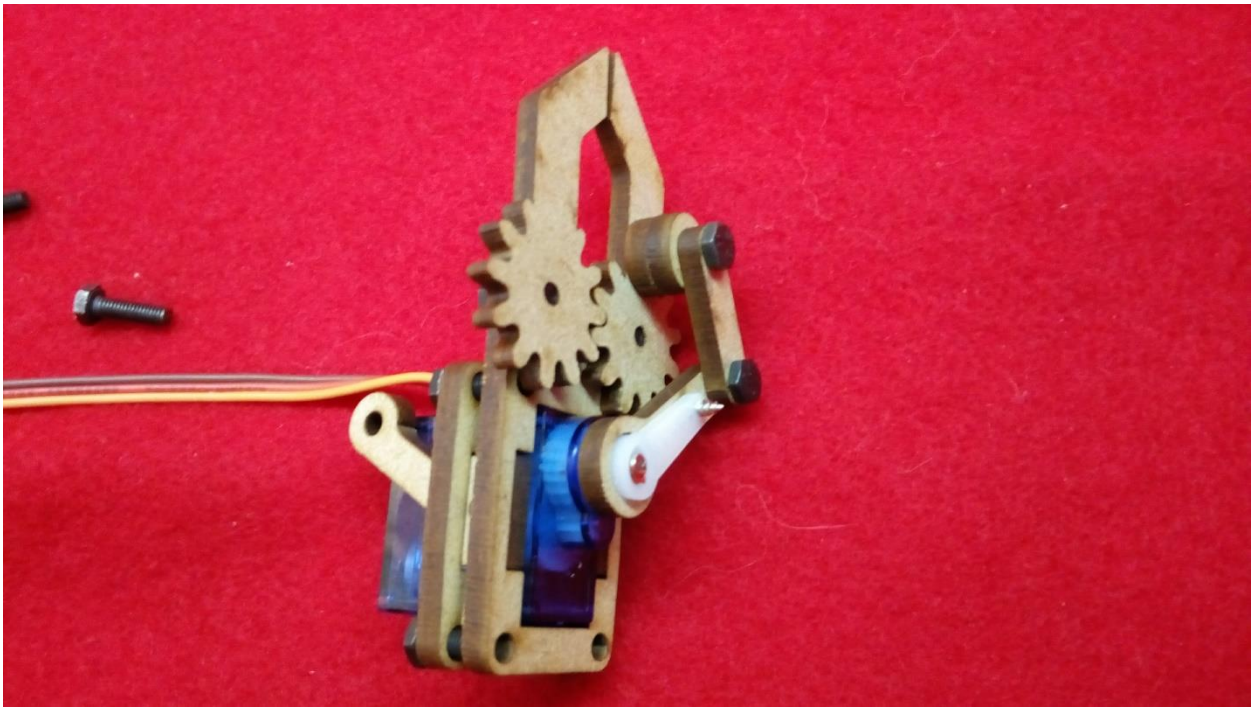




A continuación, conecte la pieza blanca del servomotor como se ve en la imagen y sujétela a la palanca con un tornillo de 6mm.

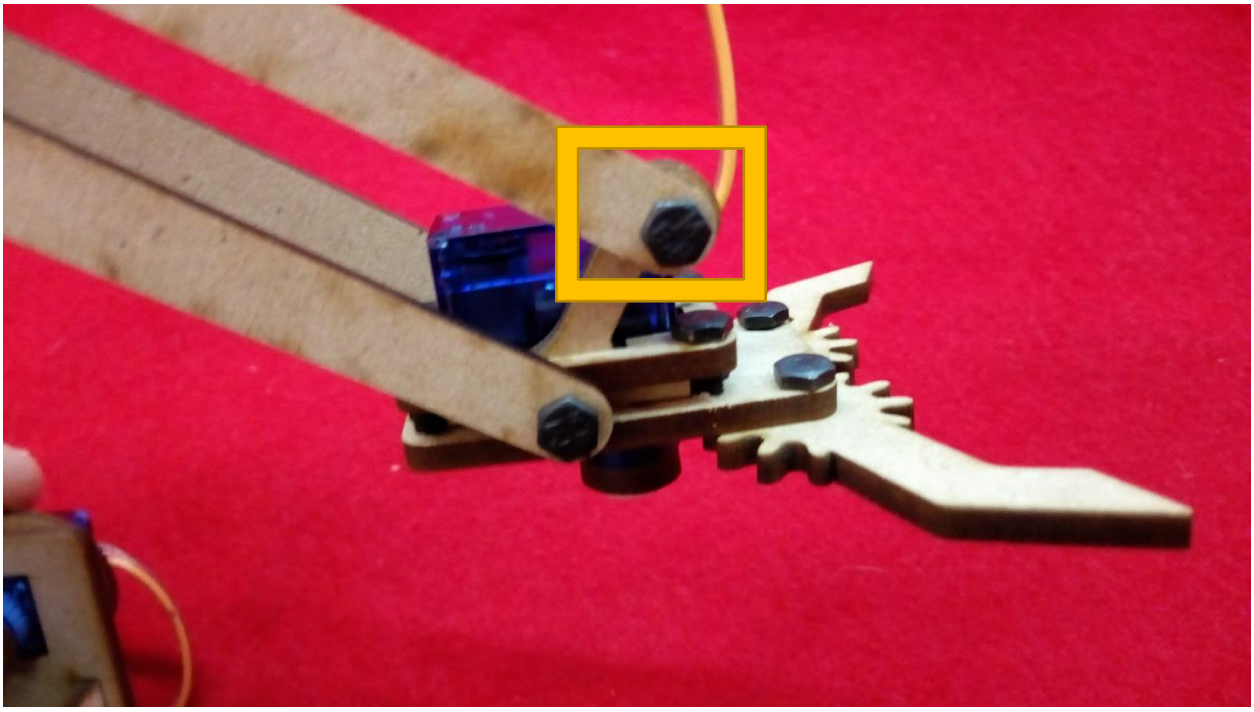


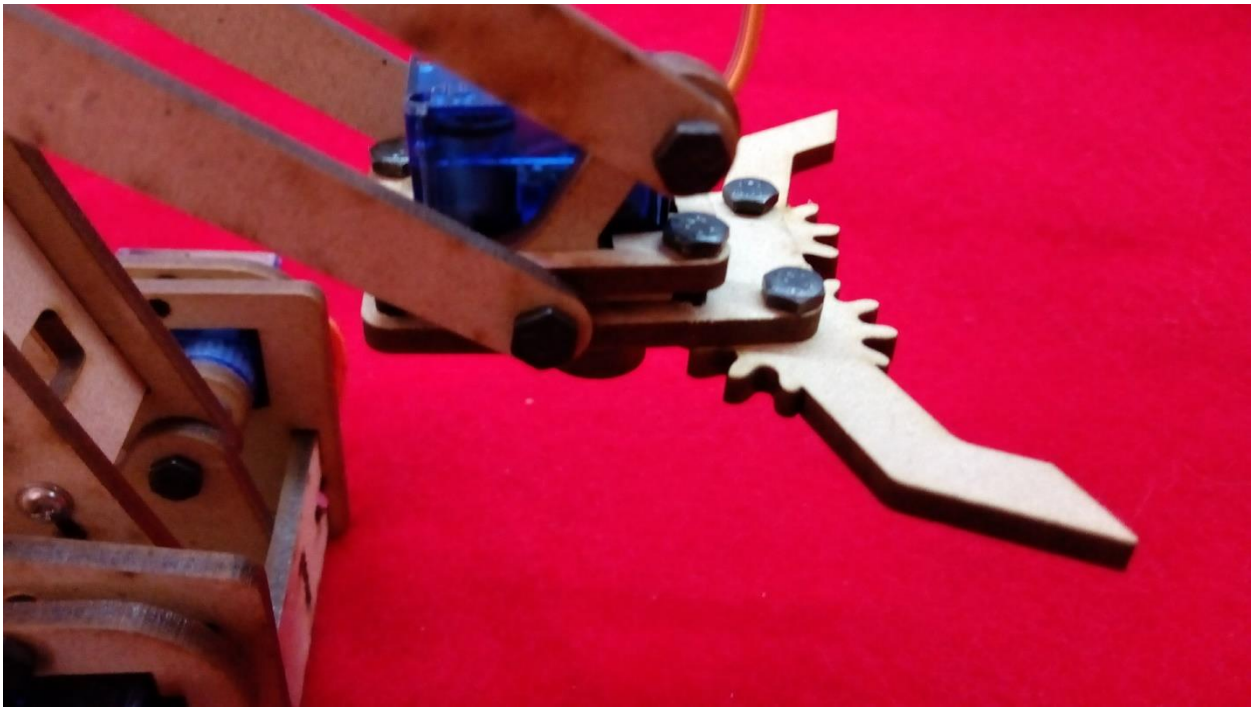
Ahora use las dos expansiones y un tornillo de 12mm para sujetar la pieza anterior con las pinzas, como se ve en la imagen, debe de ser móvil, sujete la pieza al engranaje del servomotor junto con el tornillo pequeño del servomotor.

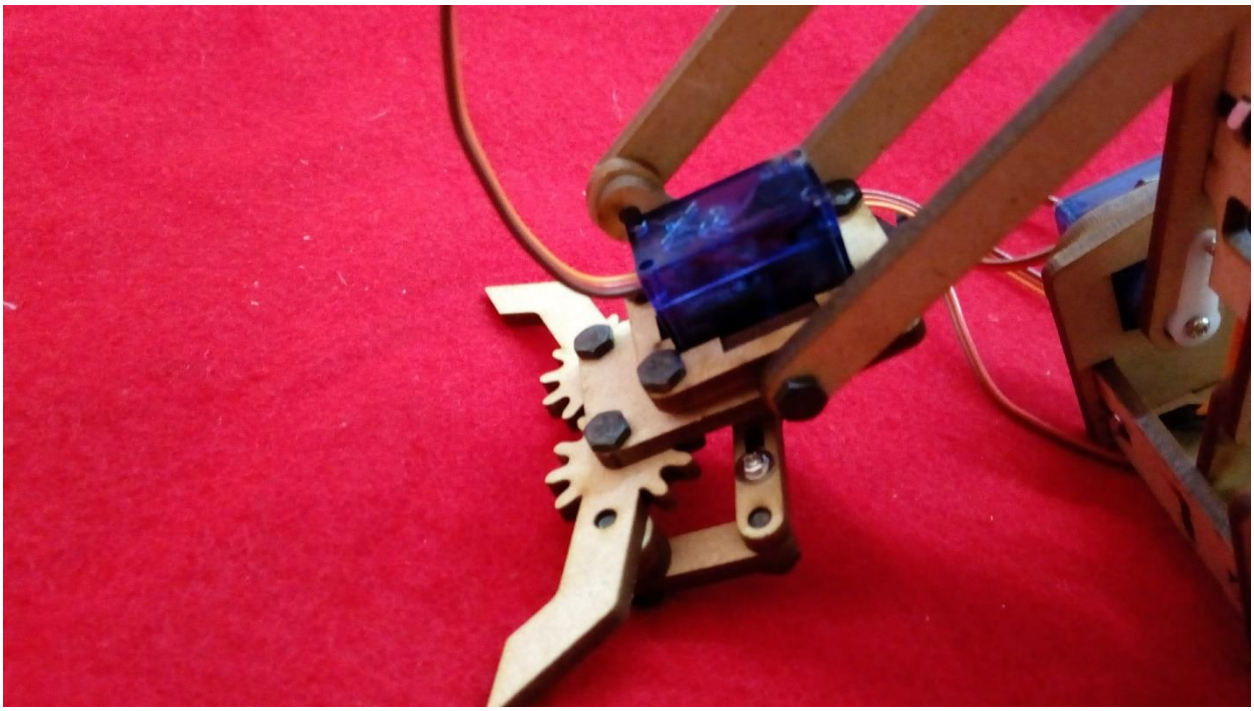


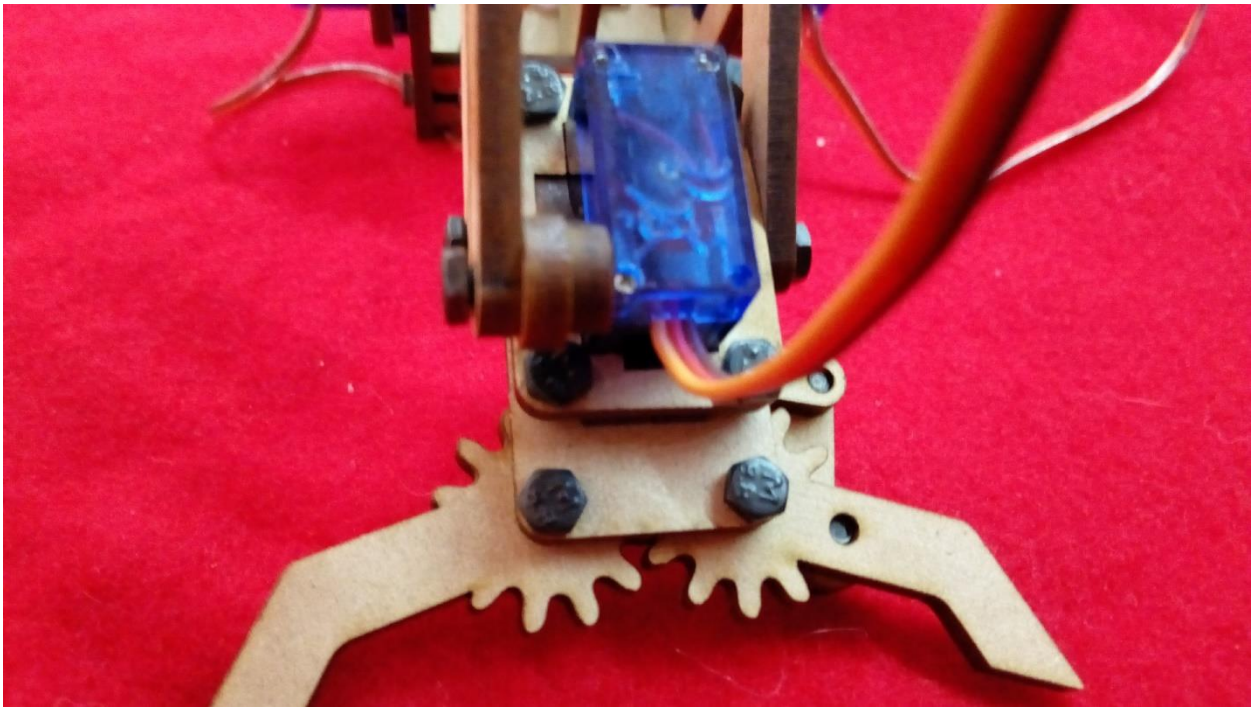
Bien, ahora sujetamos la mano a las dos palancas bajas de ambos lados con los tornillos de 8mm, usamos una expansión y el tornillo de 10mm para sujetar la tercera palanca (la más alta – marco amarillo) y estar listo, al final se debe atornillar un poco más el collar del servomotor de la mano, pero recuerde que debe

de ser una parte móvil. Con esto ha terminado de armar el brazo, ahora prosigue la programación y el circuito.







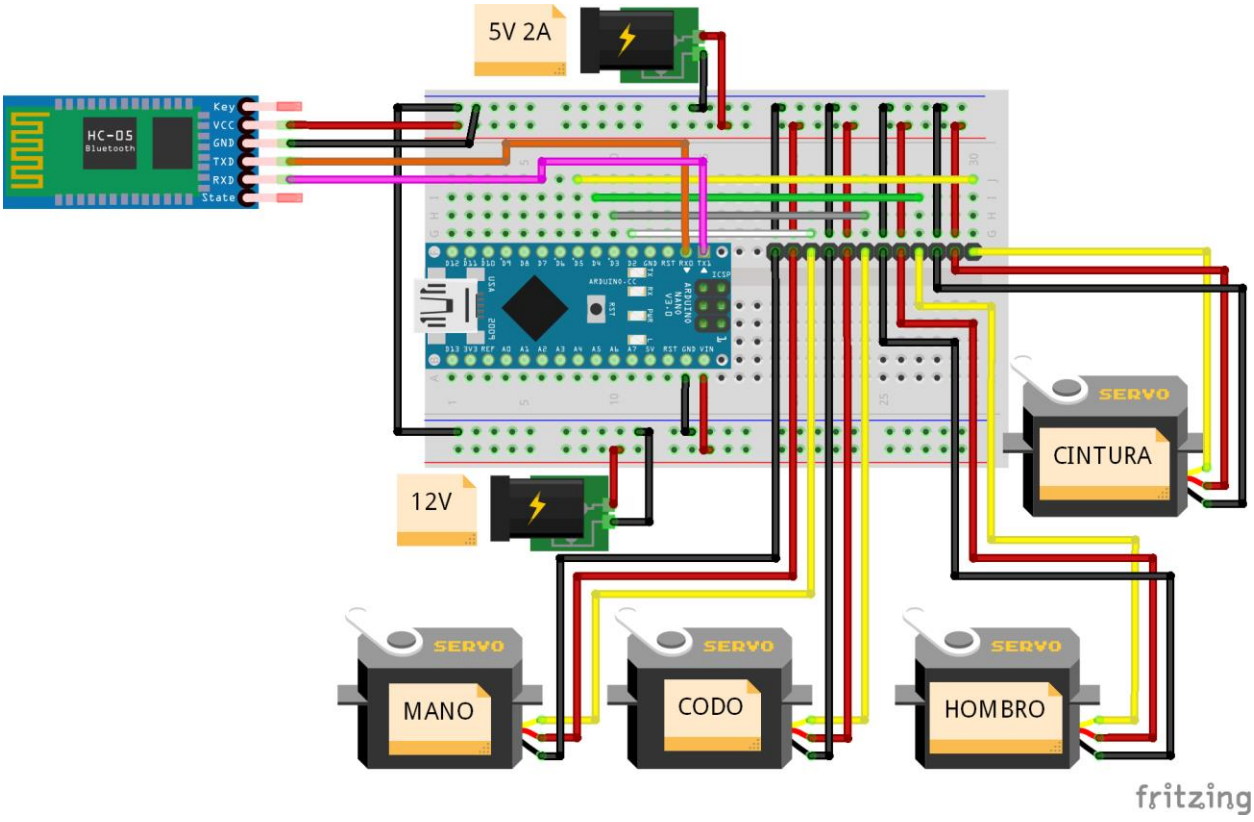


5.1.8 CIRCUITO DEL BRAZO

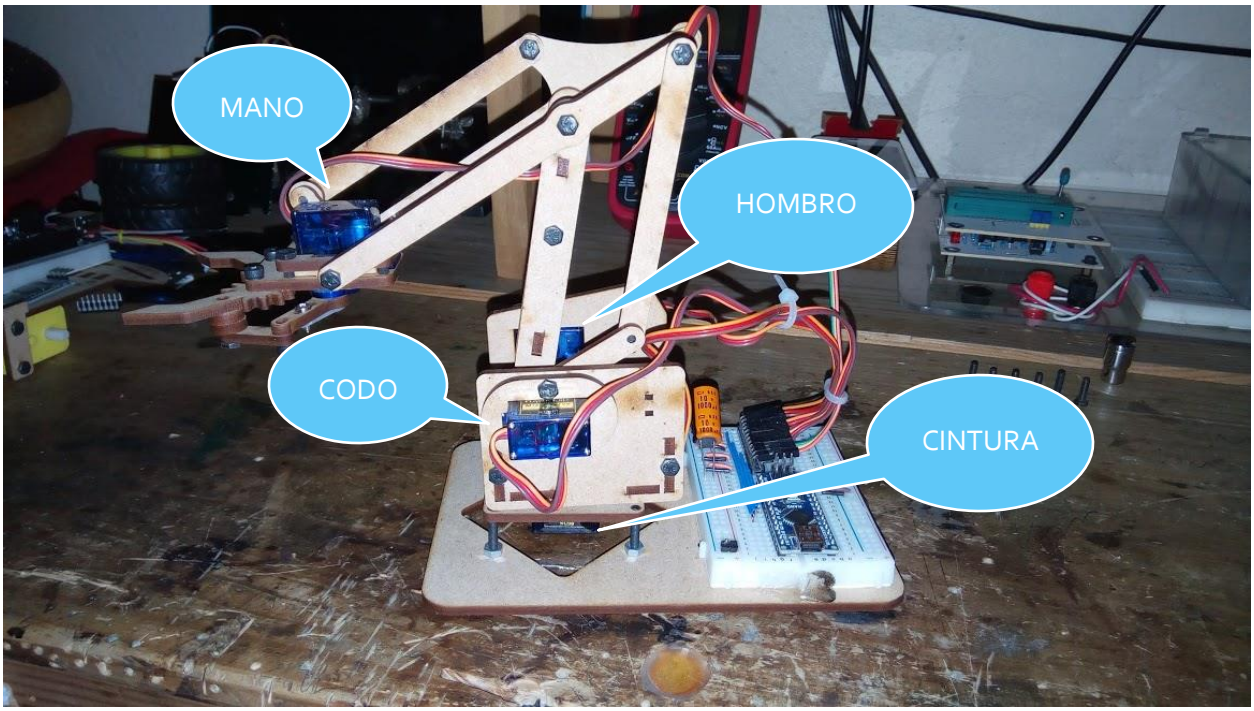
En esta parte se mostrará cómo armar el circuito del brazo, recuerde que ya se mencionó que si se usaba más de un servomotor se debería emplear una fuente externa, se recomienda una fuente para computadora ya que esta tiene salidas de 5 y 12 voltios. A continuación se lista el material para realizar el circuito, en este caso se usará un Arduino nano, pero puede usar un Arduino uno, mega, etc.

MATERIALES
Fuente de 12 y 5V a 2ª
Bluetooth HC-05 o HC-06
4 Servomotores MG90S o SG90
1 Arduino NANO o UNO
Protoboard

Circuito:



Como pueden ver en el circuito, cada servomotor tiene un nombre que concuerda con la siguiente imagen:



Código:

```
//Incluir libreria para los servos
//Completa el codigo.....

Servo mano;
/**
 * Aqui define los demas servos.
 * ...
 * ...
 */

void setup() {
 /**
  * Aqui carga el serial a 9600 misma velocidad
  * que el modulo bluetooth
  */

  mano.attach(2);
  /**
   * Carga el pin de los demas servos
   * ...
   * ...
   */
  // Se inicializan los servos a grados ya calculados usa write
```

```
  /**
   * Cargar incializacion aqui
   * mano a 80 grados
   * codo a 80 grados
   * hombro a 80 grados
   * contura a 83 grados
   */
}

void loop() {
  // Usamos un while en vez de if para saber si hay datos en el serial
  while (//Aqui pon la condicion para saber si hay datos) {
    int manoL = Serial.parseInt();
    int codoL = Serial.parseInt();
    int hombroL = Serial.parseInt();
    int cinturaL = Serial.parseInt();
    if (Serial.read() == '\n') {
    // Usa write para los servos y pasa cada variable escrita anterior
    // como parametro a la funcion write a cada servo respectivamente
      /**
       * Aqui va el codigo
       * ...
       * ...
       */
    }
  }
}
```

Explicación:

Como puede observar el código tiene comentarios con las instrucciones que usted debe de seguir, o sea debe de borrar el comentario y poner lo que éste solicita, por ejemplo; el primer comentario dice “Incluir librería para los servos” y abajo dice “Completa el código….”, debe borrar el comentario “Completa el código….” y escribir la librería para controlar los servomotores y así sucesivamente para lo demás que se pide.

Esto debe ser muy fácil ya que se vio en control y uso de motores, lo demás que se pide se vio en comunicación con Arduino.

Una vez finalizado el código debe de compilarlo y cargarlo al Arduino, recuerde que el módulo bluetooth debe de estar desconectado al momento de cargar el código, una vez cargado ya puede conectar el módulo bluetooth, usando la aplicación mencionada en el libro debe conectarse con el bluetooth y podrá controlar el servomotor con su teléfono móvil.

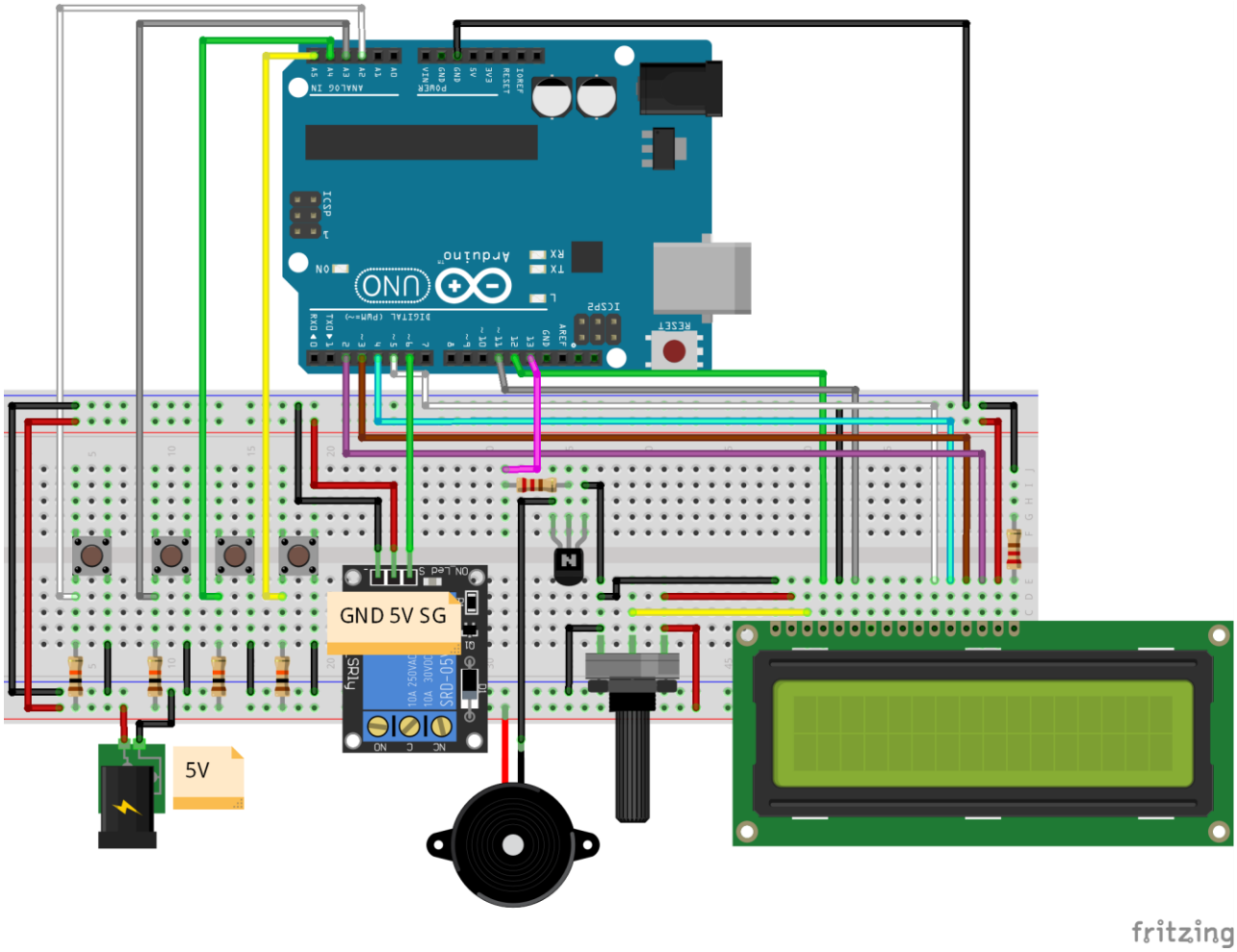
Como ve, con unas cuantas líneas de código y un poco de imaginación puede crear proyectos sumamente elaborados, como el siguiente que será un temporizador.

5.2 TEMPORIZADOR CON ARDUINO

En este proyecto se creará un temporizador, con esto podemos controlar el tiempo en que permanece prendido algún aparato eléctrico como una insoladora, un foco, televisión, etc., ya que usaremos un relevador para poder controlar grandes cargas, aquí se empleará lo visto en control y manejo de cargas, sensores digitales y mostrar datos usando una LCD.

MATERIALES
4 Pulsadores de 4 pines
4 Resistencias de 10k
1 LCD 16x2
1 Arduino
1 Relevador
1 Buzzer
1 Transistor 2N3904
1 Resistencia de 4.7k
Jumpers

Circuito:



Como podrá observar, en el circuito se han conectado cuatro pulsadores en modo pull-up a los pines desde A5 a A2, también se ha conectado un relevador al pin 6 y un buzzer al pin 7 usando un transistor tipo npn para usarlo como amplificador, y por supuesto una LCD. Las conexiones son muy sencillas y fáciles de hacer, debe hacer las conexiones con mucha precaución, una vez terminado de armar el circuito no conecte la fuente de alimentación, revise sus conexiones.

Código:

Ahora usted debe de completar el código, recuerde que tendrá los comentarios con pistas, estos comentarios se deben de borrar y poner en su lugar lo que se pide en el comentario, esto será muy fácil si ha estudiado los temas anteriores.

```
//Libreria del display LCD
//Incluye la libreria de la LCD

//Define el relevador como "RELAY" que esta conectado al pin 6
//Define la bocina como "buzzer" que esta conectado al pin 13
//Define el boton de las horas como "buth" que esta conectado al pin
A5
//Define el boton de los minutos como "butm" que esta conectado al pin
A4
//Define el boton de los segundos como "buts" que esta conectado al
pin A3
//Define el boton de arranque como "start" que esta conectado al pin
A2

LiquidCrystal lcd(//Escribe los pines de la lcd); //Inicializamos la
libreria de la lcd con el numero de los pines a utilizar

/**
 * Borra este comentario y crea 5 variables del tipo entero.
 * La primera variable se debe de llamar "ahoras" y dale un valor de
0.
 * La segunda variable se debe de llamar "aminutos" y dale un valor de
0.
 * La tercera variable se debe de llamar "asegundos" y dale un valor
de 0.
 * La cuarta variable se debe de llamar "segundostotal" y dale un
valor de 0.
 * La quinta variable se debe de llamar "msg" y dale un valor de 0.
 */

int empieza = 1;        // Variable para almacenaje del pulsador de
arranque

/**
 * Borra este comentario y crea 3 variables del tipo entero.
```

```
 * La primera variable se debe de llamar "varbuth" y dale un valor de
0.
 * La segunda variable se debe de llamar "varbutm" y dale un valor de
0.
 * La tercera variable se debe de llamar "varbuts" y dale un valor de
0.
 */

void setup(){
  /**
   * Borra este comentario he incializa la lcd
   */

  /**
   * Borra este comentario he incializa los pines como INPUT o OUTPUT
   * dependiendo de que accion hara cada pin, se da como regalo la
   * inicializacion del RELAY
   */
  pinMode(RELAY, OUTPUT);

  msg = 0;                  //Barrera del mensaje de bienvenida
  empieza = 1;            //Barrera de arranque

  varbuth = 1;              //Barrera de horas
  varbutm = 1;              //Barrera de minutos
  varbuts = 1;              //Barrera de segundos
}

void loop()
{
  if (msg == 0)           //Mostramos el mensaje de bienvenida solo una
vez
  {
    /**
     * Borra este comentario y escribe lo que se te pide:
     * Estibe un mensaje por la lcd en la fila 0 y columna 0 que diga
"Temporizador"
     * Escribe un mensaje por la lcd en la fila 1 y columan 1 que diga
"INSOLADORA" o para que aparo la usan
     * Pon un delay de 2500
     * Iguala la variable msg a 1
     * Limpia la lcd con clear();
     */
  }

  //-------------------------------------------------------------------
------------------------------
```

```
  // LECTURA DE LOS BOTONES Y ELECCIÓN DEL TIEMPO, NO SALE DEL BUCLE
HASTA PULSAR
  // EL BOTON DE ARRANQUE
  //-----------------------------------------------------------------
-------------------------------

  do{

    /**
     * Borra este comentario y asigna a las siguientes variables la
lectura de cada pulsado
     * A la variable "varbuth" asigna el valor del pulsador "buth",
recuerda digitalRead
     * A la variable "varbutm" asigna el valor del pulsador "butm",
recuerda digitalRead
     * A la variable "varbuts" asigna el valor del pulsador "buts",
recuerda digitalRead
     */

    if (varbuth == 0)             //Si el boton ha sido pulsado,
aumentamos las horas en una unidad
    {
      /**
       * Borra este comentario y escribe lo que se te pide:
       * A la variable "ahoras" debes sumarle uno (Recuerda el
capitulo uno OPERADORES DE ASIGNACION)
       * Escribe un delay de 250
       */
    }

    if (varbutm == 0)           //Si el boton ha sido pulsado,
aumentamos los minutos en una unidad
    {
      /**
       * Borra este comentario y escribe lo que se te pide:
       * A la variable "aminutos" debes sumarle uno (Recuerda el
capitulo uno OPERADORES DE ASIGNACION)
       * Escribe un delay de 250
       */
    }

    if (varbuts == 0)           //Si el boton ha sido pulsado,
aumentamos los segundos en una unidad
    {
      /**
       * Borra este comentario y escribe lo que se te pide:
```

```
       * A la variable "asegundos" debes sumarle uno (Recuerda el
capitulo uno OPERADORES DE ASIGNACION)
       * Escribe un delay de 250
       */
    }

    /**
     * Borra este comentario y escribe lo que se te pide
     * En la columna 0 y fila 0 de la lcd escribe el mensaje "Elige el
tiempo"
     */

    //Se regala el siguiente codigo...
    lcd.setCursor(4, 1);

    if (ahoras < 10) lcd.print("0");    // Si las horas son menor que
10, pone un "0" delante.
    lcd.print(ahoras);                  // Sin este codigo, se muestra
asi: H:M:S  (1:M:S)
    lcd.print(":");

    if (aminutos < 10) lcd.print("0");  // Si los minutos son menor
que 10, pone un "0" delante.
    lcd.print(aminutos);                // Sin este codigo, se muestra
asi: H:M:S  (H:1:S)

    lcd.print(":");
    if (asegundos < 10) lcd.print("0"); // Si los segundos son menor
que 10, pone un "0" delante.
    lcd.print(asegundos);               // Sin este codigo, se muestra
asi: H:M:S  (H:M:1)

    /**
     * Borra este comentario y escribe lo que se te pide
     * A la variable "empieza" asgina el valor del pulsador "start",
recuerda digitalRead
     */

    if (empieza == 0)               //Si el boton de arranque, fue
pulsado...
    {
      segundostotal = asegundos + (aminutos * 60) + (ahoras * 60 *
60);  //Convierte el tiempo elegido en segundos!!
    }

  } while (empieza != 0); // Se repite el menu de elegir tiempo hasta
que pulsemos el boton de arranque.
```

```
  //-----------------------------------------------------------------
------------------------------
  // UNA VEZ PULSADO EL BOTON DE ARRANQUE Y ACUMULADO EL TIEMPO, ENTRA
EN EL SIGUIENTE WHILE
  // Y NO FINALIZA HASTA TERMINAR LA CUENTA.
  //-----------------------------------------------------------------
------------------------------

  while (segundostotal > 0)
  {
    /**
     * Borra este comentario y escribe lo que se te pide
     * Crea un delay de 1 segundo
     * Decrementa la variable "segundostotal" de uno en uno (Recuerda
el capitulo uno OPERADORES DE ASIGNACION)
     * Activa el "RELAY" osea envia un HIGH
     */

    // Codigo de regalo ....
    ahoras = ((segundostotal / 60) / 60);  //Convertimos los segundos
totales en horas
    aminutos = (segundostotal / 60) % 60;  //Convertimos los segundos
totales en minutos
    asegundos = segundostotal % 60;        //Convertimos los segundos
totales en periodos de 60 segundos

    /**
     * Borra este comentario y escribe lo que se te pide
     * En la columna 0 y fila 0 de la lcd escribe el mensaje "Tiempo
restante"
     */

    // Codigo de regalo ....
    lcd.setCursor(4, 1);
    if (ahoras < 10) lcd.print("0");     // Si las horas son menor que
10, pone un "0" delante.
    lcd.print(ahoras);                   // Sin este codigo, se
muestra asi: H:M:S  (1:M:S)
    lcd.print(":");

    if (aminutos < 10) lcd.print("0");   // Si los minutos son menor
que 10, pone un "0" delante.
    lcd.print(aminutos);                 // Sin este codigo, se
muestra asi: H:M:S  (H:1:S)

    lcd.print(":");
```

```
    if (asegundos < 10) lcd.print("0");  // si el valor de segundo
esta por debajo de 9 (unidad) antepone un cero
    lcd.print(asegundos);                // Sin este codigo, se
muestra asi: H:M:S  (H:M:1)


    /**
     * Borra este comentario y escribe lo que se te pide
     * Crea un if con la condicion de que si "segundostotal" es igual
a 0
     * Dentro del if crea un while infinito eso se hace con while(1)
     * Dentro del while debes de escribir:
     * * Limpia la lcd con clear()
     * * En la columna 5 y fila 0 de la lcd escribe el mensaje
"Tiempo"
     * * En la columna 3 y fila 1 de la lcd escribe el mensaje
"Finalizado"
     * * Apaga el "RELAY" osea envia un LOW
     * * Crea un tipo blink con el "buzzer" con un delay de 200
milisegundos
     */
  }
}
```

Explicación:

El código es bastante amplio y “complejo”, pero muy fácil de entender, tenemos cuatro pulsadores los cuales están conectados en los pines desde el A5 al A2, estos pulsadores sirven para aumentar las horas, minutos y segundos, que será el tiempo en que estará activado nuestro relevador, esto se hará mediante los **if** que se aprecian en el código, lo que hace el pulsador de inicio es iniciar el temporizador y esto hace que comience a descontar el tiempo rompiendo la condición **do while**.

Al salir del **do while** entra a un **while** que se seguirá ejecutando siempre y cuando la variable **segundostotal** sea mayor que cero, este tiempo se calcula con operaciones matemáticas que se pueden apreciar en el código.

Después como se ve en el código, por la lcd mostramos los mensajes y el tiempo que falta para que termine, al final el relevador se desactivará y el buzzer se activará, así sonara una alarma indicándonos que ya se acabó el tiempo.

Como puede ver es sumamente simple hacer proyectos “complejos” con Arduino, lo único que se tiene que hacer es poner en práctica lo aprendido con mucha paciencia y dejar correr la imaginación.

CONCLUSIÓN

Como se pudo dar cuenta, al leer este libro no es difícil aprender a usar Arduino y crear proyectos con él, simplemente es cosa de leer las hojas de datos de los actuadores que usted use y planear la idea que usted tenga, por ejemplo si usted quiere crear una incubadora y debe monitorear la temperatura y humedad, simplemente necesitará de un sensor para hacer esas mediciones y para mostrar esos datos una LCD, y así sucesivamente con todos sus proyectos.

Como se comentó reiteradamente a través de este libro, esto es la base del aprendizaje pues existen innumerables cosas por aprender, lenguajes de programación, uso de módulos, actuadores, etc. que se deben de aprender y claro esto se le deja al lector para que adquiera un conocimiento sólido para que pueda desarrollar junto con Arduino proyectos más elaborados e incluso más allá en plataformas de desarrollo, como PIC, ARM, etc.

Como consejo al lector, no debe de abusar del copy/paste haciendo uso del código que se encuentra en la red, recuerde que este es una guía de ayuda, por si se tiene problemas al desarrollar el proyecto, si usted abusa del copy/paste no estará aprendiendo nada y cuando quiera desarrollar un proyecto no lo podrá hacer o tendrá miles de líneas de código que no sabrá que es lo que hacen, por eso es bueno que usted lo desarrolle por sí solo. Hay millones de foros de ayuda en la red, puede apoyarse en ellos al tener una duda y así lograr ser un desarrollador destacado.

Atte.: Misael Saenz Flores