




BRAZO INMOOV CONTROLADO POR GUANTE BLUETOOTH

PABLO FERRER RUFES

S21ME

18/06/2018



Contenido

1. Introducción	2
2. Material.....	2
2.1. Guante sensor	2
2.2. Mano robot	3
3. Mecánica	4
4. Electrónica.....	6
4.1. Mano Robot	6
4.2. Guante sensor	7
5. Programación	8
5.1. Código del guante sensor	10
5.2. Código Mano-Robot	11
6. Incidencias.....	13
6.1. Configuración de los módulos bluetooth.	13
6.2. Montaje de la mano robot	13
6.3. Código del programa	13
6.4. Identificación de servomotores.....	14
6.5. Alimentación Arduino Uno y servomotores	14
6.6. Calibrar servomotores.....	15

1. Introducción

En esta práctica vamos a realizar una mano robot, fabricada mediante impresoras 3D, la cual se moverá imitando el movimiento de nuestra mano. Esto es posible gracias a un guante que nos pondremos en la mano, el cual lleva un sensor en cada uno de los dedos, los cuales recogen la información correspondiente y la envían a la mano robot, para mover cada uno de los dedos independientemente mediante servomotores.

La comunicación entre el guante sensor y la mano robot se realizará mediante conexión bluetooth. Debido a esto, debemos tener dos microcontroladores. Nosotros usaremos un Arduino UNO para el control de los servomotores de la mano robot y un Arduino NANO para la lectura de los sensores del guante.

La realización del proyecto la dividiremos en tres partes: mecánica, electrónica y programación.

2. Material

Para el material necesario vamos a diferenciar entre el guante sensor y la mano robot:

Además, necesitaremos todas las herramientas correspondientes para el correcto montaje y la manipulación del proyecto, como pudiera ser un soldador y estaño, destornilladores, pinzas, alicates, pistola de silicona, tijeras...

2.1. Guante sensor

En el guante sensor necesitaremos de los siguientes materiales:

- Guante
- Arduino NANO
- Sensor de flexión x 5
- Resistencia 27k Ω x 5
- Bluetooth HC-05
- Condensador 47 μ F
- Placa perforada
- Conectores
- Cables
- Bridas
- Batería 7,4V

2.2. Mano robot

Para la mano, necesitaremos disponer de una impresora 3D para realizar la impresión de los componentes de la mano.

El diseño de esta mano es de InMoov y podemos encontrarlo en su página web www.inmoov.fr , en la cual podemos encontrar el diseño completo tanto de la mano como todas las instrucciones correspondientes para el montaje. Además de diseños de otras partes del cuerpo como podrían ser el brazo, hombro o la cabeza.

El link correspondiente a la mano, y por tanto el que nosotros seguiremos es el siguiente:

<http://inmoov.fr/hand-and-forarm/>

Además, necesitamos de los siguientes materiales:

- Arduino Uno
- Bluetooth HC-05
- Servomotor MG 996R x 5
- Placa perforada de doble cara
- Condensador 47 μ F x 3
- Conectores
- Hilo de pescar
- Pegamento plástico
- Batería 7,4V x 2
- Tornillería métrica 3

3. Mecánica

La parte más compleja de la mecánica es el montaje de la mano robótica, ya que esta se compone de muchas partes y algunas muy pequeñas.

Tras la impresión de las partes, es necesario quitar todos los soportes e imperfecciones que pudieran tener. Es aconsejable tener todas las piezas agrupadas e identificadas para facilitar el montaje.

La forma más adecuada de realizar el montaje de la mano es seguir las instrucciones de la página del diseñador.

Un dato importante a tener en cuenta es que los diseños 3D no poseen tolerancia, esto quiere decir que todas las piezas tienen un exceso de material debido al grosor del hilo utilizado para la impresión, por lo que debemos limar todas las piezas para que encajen perfectamente y las partes móviles, como pueden ser las uniones de los dedos, deslicen con facilidad.

La unión de las piezas las realizaremos con el pegamento plástico, dejándolo un tiempo para que el pegamento se seque y la unión quede fuerte.

Las articulaciones de los dedos (falanges y nudillos) hemos optado por insertar un hilo metálico, el cual es el interior de un cable rígido.

Una vez las partes principales de la mano están montadas, se insertan los motores en su soporte y se atornillan a estos para que queden fijos.

Tras esto se insertan los hilos de pescar por las partes correspondientes de cada dedo, y dejando una longitud mayor a la necesaria por si hubiera algún problema no tener que volver a hilar completamente.

Es importante no pegar la punta de los dedos hasta el final, ya que sino no tendremos forma de fijar los hilos al extremo del dedo. Una vez la mano tenga todos los hilos correspondientes (2 por dedo), procedemos a insertar estos por la muñeca, y posteriormente uniremos la mano al brazo. Si realizamos estos pasos al revés, la complicación sería mucho mayor a la hora de pasar los hilos por la parte de la muñeca.

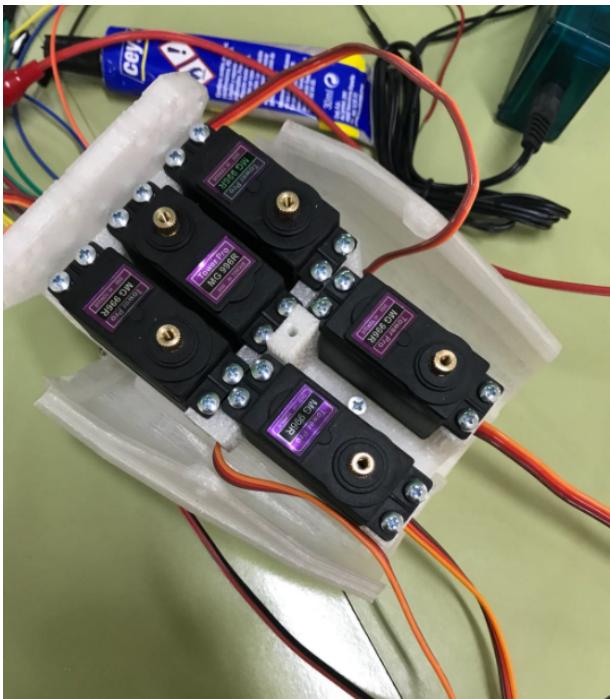
Una vez introducidos los hilos y juntada la parte de la mano a la del antebrazo, procedemos a la colocación de los hilos en las poleas, aunque no lo fijaremos aun definitivamente ya que esto habrá que modificarlo a la hora de calibrar el movimiento de los dedos.

Realizado todo esto, queda terminado el montaje de la mano robot.

Para realizar el guante sensor, simplemente hay que unir las resistencias flexibles a cada dedo.

En nuestro caso, hemos decidido hacer unas guías de papel las cuales van pegadas al dedo, y por las cuales se introducen los sensores, ya que al doblar los dedos permita un movimiento de esta y no fuerce al guante. También hemos puesto una brida para fijar más aun el movimiento del sensor y que este sea lo más sensible posible.

Luego realizaremos todas las conexiones a la paca y posteriormente la pegaremos a la parte posterior de la mano para una mayor comodidad.



4. Electrónica

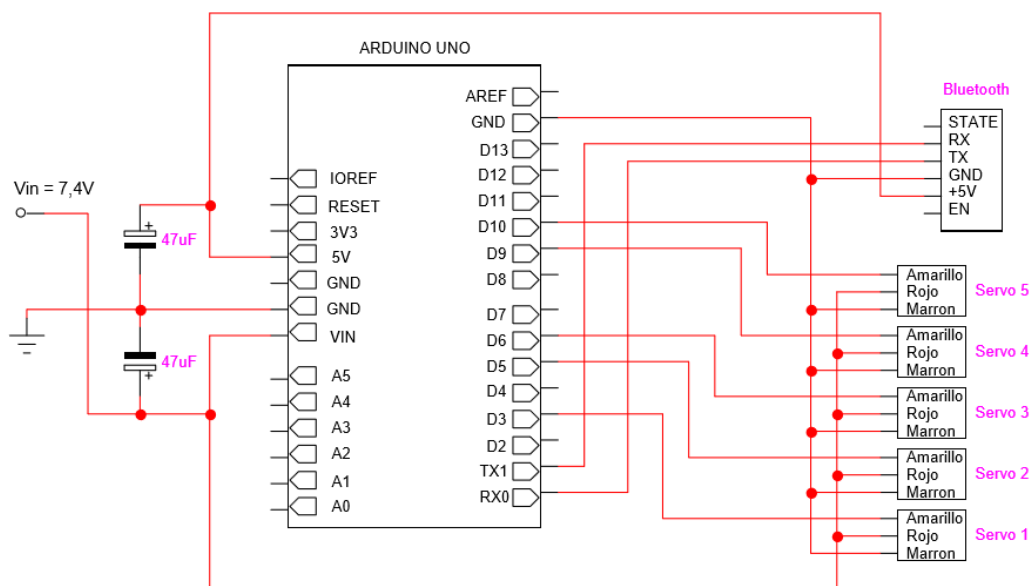
La realización de la electrónica de este proyecto es la parte más sencilla, con respecto a la programación y la mecánica. Gracias a Arduino, únicamente necesitamos realizar las conexiones adecuadas con este, excepto en el guante, en el cual hemos realizado un divisor de tensión por cada dedo para la medición de los sensores.

Por lo tanto, la placa de la mano robot es simplemente para facilitar las conexiones y colocar algunos condensadores de protección.

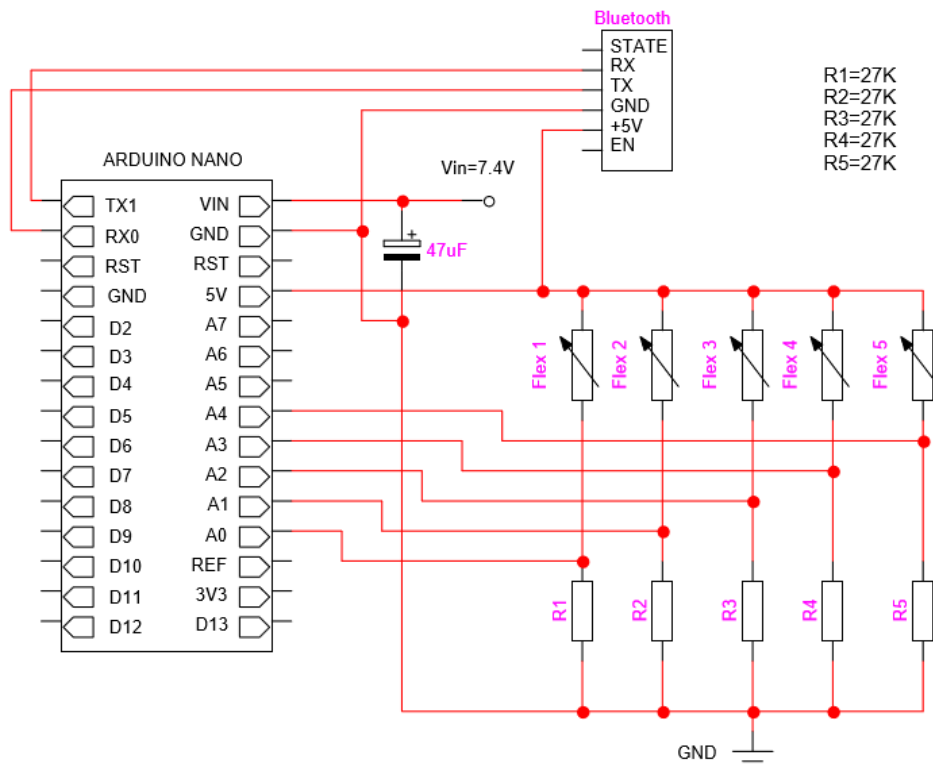
En la placa del guante, como hemos comentado antes, necesitamos realizar unos divisores de tensión con las resistencias y los sensores de flexión. Además, aquí también hemos añadido un condensador de protección y hemos soldado completamente el Arduino NANO a la placa.

Los esquemas electrónicos son los siguientes:

4.1. Mano Robot



4.2. Guante sensor



Como podemos comprobar, las conexiones son sencillas. Principalmente debemos tener en cuenta varios detalles:

- Una vez que soldemos el bluetooth al módulo Arduino no podremos volver a reconfigurar éste. Esto se debe a un error que da Arduino al subir un proyecto cuando se encuentra el RX0 conectado a otro dispositivo. Una solución es soldar unos conectores a los cuales podremos conectar o desconectar el bluetooth cuando lo necesitemos.
- Debemos tener cuidado con la alimentación. Puesto que estamos alimentando con 7,4V no debemos usar esto como la parte positiva del divisor de tensión. Arduino trabaja con valores de 0 a 5V, para los cuales devuelve un valor de entre 0 y 1023.
- Las conexiones de TX1 y RX0 deben estar invertidas con el modulo bluetooth, es decir, el TX debe estar conectado al RX y viceversa, ya que cuando uno envía el dato, el otro debe recibirlo.
TX se usa para enviar información y RX se usa para recibir esta información enviada.
- La alimentación de los servomotores debemos hacerla de manera independiente al Arduino UNO, ya que, si se alimenta el Arduino junto con los motores, produce pequeños picos de caída que produce que el Arduino se conecte y reconecte constantemente, lo que afecta a la conexión del bluetooth e impide un correcto funcionamiento.
- Siempre debemos tener todas las tierras unidas.

5. Programación

La programación de este proyecto se complica considerablemente mediante el uso de los dispositivos bluetooth, ya que no trabajamos solo con un microcontrolador, sino con dos.

Además, es necesario establecer una comunicación entre estos microcontroladores. El Arduino NANO del guante sensor debe mandar la información de cada motor (identificando cada uno de ellos para evitar que se entremezcle la información) y el Arduino UNO de la mano debe recibir esta información correctamente, identificar a que motor pertenece, realizar los ajustes necesarios para que el movimiento coincida con el del guante y enviar la señal a los motores para su movimiento.

Inicialmente, antes de empezar con la programación de los microcontroladores debemos configurar los módulos bluetooth para que se conecten entre ellos. Esto se consigue teniendo un módulo esclavo y otro maestro. El modulo del que disponemos nosotros, el HC-05, poseen la capacidad de ser configurados tanto como esclavo como maestro.

Para poder configurar estos módulos debemos realizar una serie de pasos:

- Debemos cargar en el Arduino un programa que permita comunicar el Arduino con el modulo bluetooth. Para esto se utiliza el puerto serie del Arduino (TX, RX) para enviar los comandos a través del monitor y además necesitaremos crear otro puerto serie adicional para la conexión con el bluetooth.

El código es el siguiente:

```
#include <SoftwareSerial.h>
int poti;
char c = ' ';
SoftwareSerial BTSerial (3, 2); // Puerto serie BT RX-TX

void setup() {
  Serial.begin(9600);
  Serial.println("Arduino esta listo");

  BTSerial.begin(38400);
}

void loop() {
  // Realiza la comunicacion entre Arduino y BT
  if (BTSerial.available())
    Serial.write(BTSerial.read()); // Escribe en el monitor lo que proviene del BT

  if (Serial.available()) // Cuando hay datos disponible en el monitor serie
  {
    String S = GetLine(); // Lee una linea entera hasta intro
    BTSerial.print(S); // Envia la linea completa al BT
    Serial.println("----> " + S); // Imprime en el monitor lo mismo que se envia
  }
}

// Sub-Programa que permite leer una linea entera por el puerto serie
String GetLine()
{
  String S = "";
  if (Serial.available())
  {
    char c = Serial.read(); ;
    while ( c != '\n') //Hasta que el caracter sea intro
    {
      S = S + c ;
      delay(25) ;
      c = Serial.read();
    }
    return( S + '\n' ) ;
  }
}
```

- Una vez cargado el código, y con la placa conectada, debemos conectar el modulo bluetooth a la vez que pulsamos el pequeño botón que tiene. Haciendo esto, el bluetooth entrará en modo AT, desde la cual se podrá configurar y mientras no se conectará a ningún otro dispositivo. Solo necesitamos conectar la alimentación y TX, RX del bluetooth.
- Para realizar la configuración, entramos al monitor serie del Arduino. En la parte inferior debemos escoger "Ambos NL & CR" y la velocidad "38400 baud" e introducimos los comandos AT correspondientes para la configuración que deseemos. Comenzamos escribiendo solamente AT para empezar la configuración Si recibimos un OK quiere decir que todo está correctamente. A continuación, cualquiera de estos comandos:
 - AT+NAME=<Nombre>
Cambia el nombre. Ejm: AT+NAME=Robot
 - AT+PSWD=<Pin>
Cambia el pin. Ejm: AT+PSWD=2560
 - Enviar: AT+UART=<Baud>
Cambia la velocidad de comunicación
 - Enviar: AT+ROLE=<Role>
Configurar como maestro (1) o esclavo (0)

Para conocer estos valores, debemos escribir estos mismos comandos eliminando la parte a continuación del igual junto con este:

Ejm: AT+NAME // Y nos devolvería el nombre de este.

En esta página Web podemos encontrar toda la lista de comandos AT posibles.
http://www.naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usa.html

- Una vez configuramos ambos como esclavo y maestro, debemos asegurarnos de que ambos posean el mismo pin (PSWD).
- Configuramos el maestro para que solo se conecte a la dirección especificada e introducimos la dirección de nuestro modulo esclavo.

Tras realizar estos pasos los dispositivos bluetooth quedarían completamente configurados y no sería necesario volver a repetir este proceso.

Ahora sí, pasamos a realizar la configuración de nuestro programa para la mano.

Necesitamos dos programas, uno para la mano y otro para el guante, como hemos dicho anteriormente:

5.1. Código del guante sensor

```
#include <SoftwareSerial.h>
int dato;
char datochar;
void setup()
{
    Serial.begin(9600); // Inicia la comunicacion serie a 9600bps
}

void loop()
{
    for(unsigned i=0; i<5; i++)
    {
        dato = analogRead(i);           // Lee el valor del sensor
        dato = map(dato, 0, 1023, 0, 255); // Convierte el dato a 1 byte
        datochar = dato;                 // Pasa el dato a un tipo char para enviarlo

        if (i == 0)                     // Identifica que sensor lee
        {
            Serial.write('a');           // Escribe una letra para identificarlo
            delay(5);                    // Espera para no saturar el buffer de memoria del bluetooth
            Serial.write(dato);           // Escribe el dato correspondiente al sensor
            delay(5);                    // Espera a que de tiempo de mover el motor por la otra placa
        }
        if (i == 1)                     // Vuelve a repetir el mismo proceso para los demas sensores
        {
            Serial.write('b');
            delay(5);
            Serial.write(dato);
            delay(5);
        }
        if (i == 2)
        {
            Serial.write('c');
            delay(5);
            Serial.write(dato);
            delay(5);
        }
        if (i == 3)
        {
            Serial.write('d');
            delay(5);
            Serial.write(dato);
            delay(5);
        }
        if (i == 4)
        {
            Serial.write('e');
            delay(5);
            Serial.write(dato);
            delay(5);
        }
    }
}
```

Como podemos comprobar, este programa lee la información recibida del sensor, lo identifica mediante una letra, según de que sensor se trate, y envía esta información mediante dos Bytes.

5.2. Código Mano-Robot

```
#include <SoftwareSerial.h>
#include <Servo.h>

unsigned char valor;           // Declara un caractere sin la parte negativa
int num; // declara tipo entero
Servo servol, servo2, servo3, servo4, servo5; // declara servol, servo2, servo3, servo4, servo5
char letra;                   // declara una variable tipo caracter

void setup()                  // Setup Code
{
  pinMode(3, OUTPUT);         // Inicializa pin 3 como Salida
  pinMode(5, OUTPUT);         // Inicializa pin 5 como Salida
  pinMode(6, OUTPUT);         // Inicializa pin 6 como Salida
  pinMode(9, OUTPUT);         // Inicializa pin 8 como Salida
  pinMode(10, OUTPUT);        // Inicializa pin 10 como Salida

  servol.attach(3);           // inicializa servol al pin 3
  servo2.attach(5);           // inicializa servo2 al pin 5
  servo3.attach(6);           // inicializa servo3 al pin 6
  servo4.attach(9);           // inicializa servo4 al pin 9
  servo5.attach(10);          // inicializa servo5 al pin 10

  Serial.begin(9600);         // Inicia comunicacion serial a 9600bps
}

void loop()
{
  // espera datos como "a0" "b180" or "c270"
  if(Serial.available() > 0) // si hay datos disponibles
  {
    letra = Serial.read();    // lee el primer dato del buffer
    if (letra == 'a')         // si el dato es una 'a'
    {
      while(Serial.available() < 1) // espera a recibir el siguiente dato
      {
      }
      valor = Serial.read();        // lee el valor
      Serial.println(valor);

      if(valor > 39 && valor < 110) //si el valor esta entre los rangos de la resistencia flexible
      {
        valor = map(valor, 50, 85, 50, 180); // ajusta el valor para el recorrido del servo adecuado a cada dedo
        servol.write(valor);                 // mueve el servo
      }
    }
    if (letra == 'b')           // Vuelve a repetir el proceso para los demas datos
    {
      while(Serial.available() < 1)
      {
      }
      valor = Serial.read();
      if(valor > 39 && valor < 110)
      {
        valor = map(valor, 50, 95, 100, 180);
        servo2.write(valor);
      }
    }
    if (letra == 'c')
    {
      while(Serial.available() < 1)
      {
      }
      valor = Serial.read();

      if(valor > 39 && valor < 110)
      {
        valor = map(valor, 40, 95, 30, 155);
        servo3.write(valor);
      }
    }
    if (letra == 'd')
    {
      while(Serial.available() < 1)
      {
      }
      valor = Serial.read();

      if(valor > 39 && valor < 110)
      {
        valor = map(valor, 55, 105, 30, 180);
        servo4.write(valor);
      }
    }
  }
}
```

```

if (letra == 'e')
{
  while(Serial.available() < 1)
  {
  }
  valor = Serial.read();
  if(valor > 39 && valor < 110)
  {
    valor = map(valor, 45, 85, 50, 180);
    servo5.write(valor);
  }
}
}
}

```

Vemos que este código es algo más complejo ya que además de recibir los datos debe hacerle un cambio para ajustarlo al movimiento del motor.

Primero recibe el dato, y si es una de las letras de identificación, espera a recibir el siguiente dato el cual será el valor del sensor.

Transforma el dato cambiándole los rangos para adaptarlos al movimiento del motor y por ultimo utilizamos una función de la librería servo para comunicar al servomotor en qué posición se debe poner

La función que mueve los motores acepta solamente valores con un rango de entre 0 y 180.

6. Incidencias

Los problemas encontrados principalmente son los siguientes, desde el inicio hasta el fin del montaje del proyecto.

6.1. Configuración de los módulos bluetooth.

Este paso resulto bastante complejo ya que necesita de un código en concreto que permita la comunicación entre bluetooth y Arduino. Puesto que es la primera vez que trabajaba con la plataforma Arduino no disponía de los conocimientos necesarios para configurar un puerto serie. Por lo que tuve que buscar un código ya completo para poder realizar la configuración AT de los módulos bluetooth. Tras usar el código adecuado los módulos se conectaban correctamente.

6.2. Montaje de la mano robot

Varias de las piezas que formaban la mano estaban mal impresas por lo que retraso el montaje de la mano. Fue necesario volver a imprimir estas piezas ya que es necesario que encajen perfectamente.

También había piezas que faltaban, como varios dedos de la mano, por lo que también hubo que imprimirlas a parte.

Estas piezas se pueden identificar ya que son las piezas de color negro.

6.3. Código del programa

Esta ha sido la parte que más tiempo ha costado resolver, ya que no se trata de un solo problema sino el conjunto de algunos.

Uno de los problemas era el uso del tipo de dato "char", que, aunque tiene el tamaño de un Byte (256), es un tipo de dato con valor negativo, es decir, el rango de valores va de -128 a 127. Por lo que trabajar con este dato dificulta bastante el programa, ya que el dato que se envía es de 0 a 255. Esto se puede solucionar fácilmente usando el tipo de dato "unsigned char".

Otro de los problemas del código fue la transmisión y recepción de datos. Puesto que el puerto serie solo es capaz de recibir un Byte únicamente, había que modificar todos los datos para enviar únicamente un Byte, y escoger adecuadamente el tipo de dato tanto en el código del envío como en el de recepción para asegurarnos de recibir el mismo dato que enviamos.

Tras probar distintos datos e ir comprobando con el monitor serie el dato recibido obtenemos que el tipo de dato del envío es un tipo "char" y la recepción mediante un tipo "unsigned char", así obtenemos el dato desde 0 a 255.

6.4. Identificación de servomotores

Otro problema encontrado fue como identificar cada sensor, para recibir la información correctamente. Si solamente enviamos los datos de los sensores no tenemos forma de identificar a que dedo corresponde, ya que, aunque ambos empezaran en el mismo, los dos módulos bluetooth deberían empezar la comunicación exactamente en el mismo momento que se empieza a enviar el primer dato. Cosa que es imposible debido a que el bluetooth tarda un cierto tiempo en realizar la conexión y comenzar la transmisión.

Este problema se puede solucionar mediante distintas opciones.

En nuestro caso la solución ha sido añadir un Byte de identificación justo antes del Byte de información del sensor. El Byte era simplemente un carácter, uno para cada sensor, el cual permite de esta manera identificar cada uno de ellos. Los datos enviados serán entonces dos Bytes por cada sensor, el de identificación y el de información.

La información enviada será de esta forma: "a95", "b79" o "c120".

Conociendo esto, en el código de recepción debemos buscar estos caracteres de identificación, y esperar por el siguiente dato, el cual sabemos que es el que contiene la información.

6.5. Alimentación Arduino Uno y servomotores

Inicialmente tanto el Arduino UNO como los servomotores estaban alimentados desde la misma batería. Esto funcionaba correctamente hasta que se conectaban tres o más servomotores. En ese momento el bluetooth se desconectaba continuamente.

En un principio pensaba que podía ser debido a que la batería no suministrara suficiente corriente, por lo que procedí a conectarlo a una fuente de alimentación, la cual posee corriente suficiente para alimentar todos los motores, sin embargo, el problema persistía de la misma manera.

El siguiente intento de solucionarlo fue poner condensadores en la entrada de la alimentación del Arduino, así como en la alimentación del módulo bluetooth, pero aun así seguía dando el mismo error.

Tras esto, vimos que el error se debía a que el Arduino se desconectaba debido a los picos de consumo de los servomotores, por lo que la opción para solucionarlo fue añadir otra batería, para alimentar los motores de manera independiente. Tras realizar este cambio vimos que todo funciona correctamente.

6.6. Calibrar servomotores

Por último, tenemos este problema, el cual no es un fallo, sin embargo, resulta muy tedioso.

El problema es la calibración de los motores y de ajustar el hilo a estos correctamente, los motores se calibran desde el código, cambiando el ángulo de giro para que tire más o menos del hilo de los dedos. El siguiente paso es fijar los hilos a los servomotores, los cuales deben estar bien tensados para que no se salgan de las guías al girar.

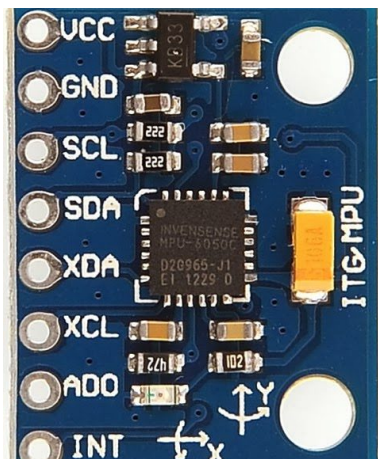
Esta última parte es más complicada, ya que es difícil tensar ambos hilos a la vez, debido a que cuando el motor gira para tensar uno, el otro se destensa provocando que se salga de las guías.

Una solución que he realizado ha sido empezar con la mano abierta y tensar el hilo que corresponde a este movimiento, a continuación, he unido el otro hilo tensado y mediante código he ajustado el movimiento de servomotor necesario para que el dedo se contraiga completamente. Y repetir este proceso para cada dedo de forma independiente, ya que el movimiento de cada uno de estos es distinto.

7. Ampliación con el sensor MPU6050

7.1. Introducción

En este apartado vamos a ampliar nuestro proyecto, añadiéndole el movimiento de giro de la muñeca, para el cual usaremos un servomotor, controlado con el sensor de giroscopio de 6 ejes MPU6050.



7.2. MPU6050

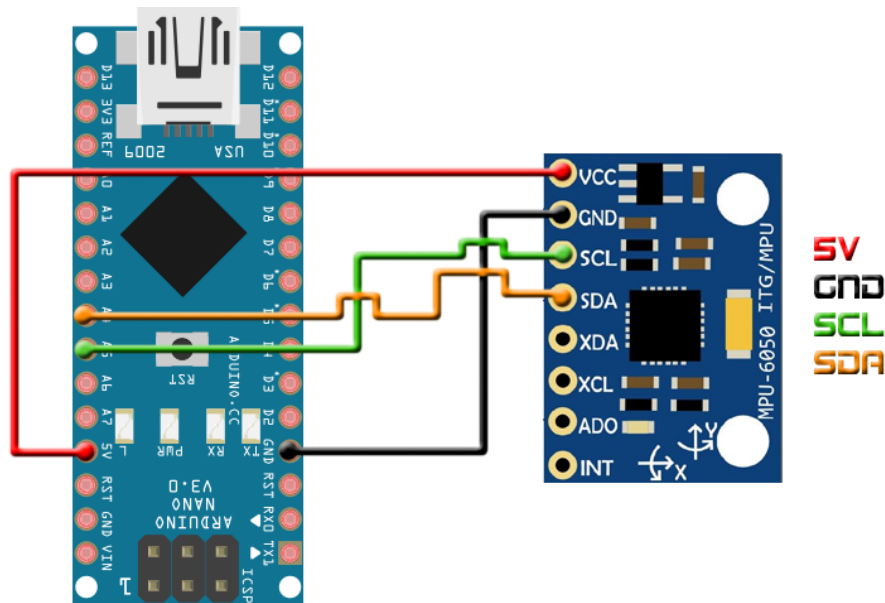
El MPU650 que vamos a utilizar es un chip de 6 dof o grados de libertad, ya que incluye un acelerómetro de 3 ejes y un giróscopo de 3 ejes.

Aunque lo que miden los sensores internos son aceleraciones lineales y angulares, el procesador interno del IMU es capaz de realizar cálculos sobre la marcha para darnos informaciones más útiles como los ángulos de inclinación con respecto a los 3 ejes principales.

En nuestro caso lo conectaremos al Arduino Nano, ya que es desde el cual tomaremos las mediciones del movimiento de la mano. Además, Arduino Nano posee 8 puertos analógicos, a diferencia de Arduino Uno, el cual solo posee 6, y por lo tanto no tendríamos disponibles puertos suficientes para medir los 5 sensores de los dedos.

El MPU se comunica mediante la interfaz I2C, la cual utiliza un cable para el impulso de reloj (SCL) y otro para la transmisión de datos (SDA). Esta interfaz se debe conectar al Arduino Nano a los puertos analógicos A5 y A4, respectivamente.

Quedando la conexión de tal manera:



El módulo tiene un regulador de voltaje en placa de 3.3V, el cual se puede alimentar con los 5V del Arduino.

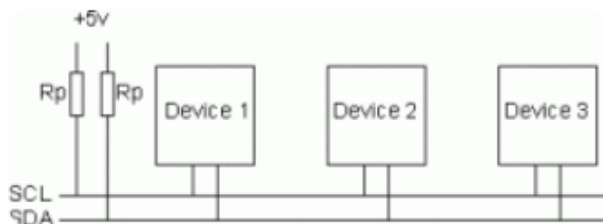
El pin ADDR internamente en el módulo tiene una resistencia a GND, por lo que, si no se conecta, la dirección por defecto será 0x68.

7.3. Interfaz I2C

El I²C es un protocolo serie para la interfaz de dos cables, que facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de "inteligencia".

Se trata de, un bus bidireccional que utiliza dos líneas, una de datos serie (SDA) y otra de reloj serie (SCL). SCL es la línea de reloj, se utiliza para sincronizar todos los datos SDA de las transferencias durante I²C bus.

Las líneas SCL y SDA están conectadas a todos los dispositivos en el I²C bus. La alimentación del sistema, debe tener una masa común, también puede haber una alimentación compartida que, se distribuye entre los distintos dispositivos.



Los dispositivos en el I²C bus son maestros o esclavos. El maestro, es siempre el dispositivo que maneja la línea de reloj SCL. Los esclavos, son los dispositivos que responden al maestro. Un esclavo no puede iniciar una transferencia a través del I²C bus, sólo un maestro puede hacer esa función. Generalmente son, varios esclavos en el I²C bus, sin embargo, normalmente hay un solo maestro. Es posible tener varios maestros, pero es inusual. Los esclavos, nunca inician una transferencia. Tanto el maestro, como el esclavo puede transferir datos a través del I²C bus, pero la transferencia siempre es controlada por el maestro.

Todas las direcciones I²C bus son de 7 bits. Esto significa que, permite hasta 112 nodos en un bus (16 de las 128 direcciones posibles están reservadas para fines especiales), ya que un número de 7bit puede estar de 0 a 127.

Debido a la escasez de direcciones, se introdujo más tarde un direccionamiento de 10 bits.

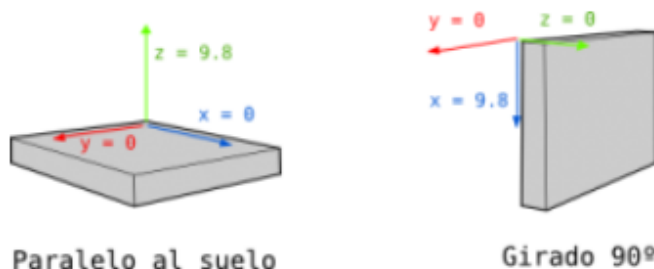
Cuando se envía la dirección, siempre se envían 8 bits. El bit extra (bit 8º) se usa para informar al esclavo si el maestro está escribiendo o leyendo de él.

Una de las propiedades del I²C es el hecho de que un microcontrolador puede controlar toda una red de circuitos integrados con sólo dos I/O-Pins (Input/Output) y un software muy simple. Aunque es más lento que los sistemas de bus más nuevos, I²C es beneficioso (debido al bajo coste) para los sistemas periféricos que no necesitan ser rápidos. A menudo es usado para la transmisión de datos de control y configuración, por ejemplo, para control de volumen, conversor de señal analógica-digital o digital-analógica con baja tasa de frecuencia de muestreo, relojes a tiempo real, pequeños espacios de memoria o conmutadores bidireccionales y multiplexores. Incluso los sensores electrónicos integran con frecuencia un convertidor analógico-digital con un I²C.

7.4. Acelerómetro

El acelerómetro mide la aceleración. La aceleración puede expresarse en 3 ejes: X, Y y Z, las tres dimensiones del espacio. Por ejemplo, si mueves la IMU hacia arriba, el eje Z marcará un cierto valor. Si es hacia delante, marcará el eje X, etc.

La gravedad de la Tierra tiene una aceleración de aprox. 9.8 m/s^2 , perpendicular al suelo como es lógico. Así pues, la IMU también detecta la aceleración de la gravedad terrestre.



Gracias a la gravedad terrestre puedes usar las lecturas del acelerómetro para saber cuál es el ángulo de inclinación respecto al eje X o eje Y. Supongamos que la IMU esté perfectamente alineada con el suelo. Entonces, como puedes ver en la imagen, el eje Z marcará 9.8, y los otros dos ejes marcarán 0. Ahora supongamos que giramos la IMU 90 grados. Ahora es el eje X el que está perpendicular al suelo, por lo tanto, marcará la aceleración de la gravedad.

Si sabemos que la gravedad es 9.8 m/s^2 , y sabemos que mide los tres ejes del acelerómetro, por trigonometría es posible calcular el ángulo de inclinación de la IMU. Una buena fórmula para calcular el ángulo es:

$$\begin{aligned} \text{AnguloY} &= \text{atan} \left(\frac{x}{\sqrt{y^2 + z^2}} \right) \\ \text{AnguloX} &= \text{atan} \left(\frac{y}{\sqrt{x^2 + z^2}} \right) \end{aligned}$$

Dado que el ángulo se calcula a partir de la gravedad, no es posible calcular el ángulo Z con esta fórmula ni con ninguna otra. Para hacerlo se necesita otro componente: el magnetómetro, que es un tipo de brújula digital. El MPU-6050 no lleva, y por tanto nunca podrá calcular con precisión el ángulo Z. Por este motivo, para calcular el ángulo usamos el giroscopio.

7.5. Giroscopio

El giroscopio mide la velocidad angular. La velocidad angular es el número de grados que se gira en un segundo.

Si sabemos el ángulo inicial de la IMU, podemos sumarle el valor que marca el giroscopio para saber el nuevo ángulo a cada momento. Supongamos que iniciamos la IMU a 0° . Si el giroscopio realiza una medida cada segundo, y marca 3 en el eje X, tendremos el ángulo con esta sencilla fórmula:

$$\text{AnguloY} = \text{AnguloY Anterior} + \text{GiroscopioY} \cdot \Delta t$$

Dónde Δt es el tiempo que transcurre cada vez que se calcula esta fórmula, AnguloY Anterior es el ángulo calculado la última vez que se llamó esta fórmula y GiroscopioY es la lectura del ángulo Y del giroscopio.

Y lo mismo pasa con los ejes X, Z. Sólo que se suele ignorar el eje Z, puesto que al no poder calcular un ángulo Z con el Acelerómetro, no se puede aplicar un Filtro Complementario para el eje Z.

A diferencia del acelerómetro, da las medidas con mucha precisión. Pero al realizar los cálculos del ángulo es inevitable que se produzca un pequeño error, que con el tiempo va acumulándose hasta que cualquier similitud con la realidad es pura coincidencia. Esto en inglés se llama drift.

Existen varias maneras de eliminar este ruido, entre las que encontramos distintos filtros como pueden ser el Filtro de Kalman, o el Filtro Complementario.

Éste último combina las mediciones del acelerómetro y del giroscopio para mostrar una medición más precisa. Pero en nuestro caso, ya que no necesitamos mediciones tan exactas, sino que el movimiento del motor sea similar al giro de nuestra mano, hemos optado por otro método.

Hemos añadido un botón el cual ira en la mano, que permite mover la mano mientras mantenemos el robot completamente quieto, lo que nos permite reposicionar nuestra mano hasta la posición deseada.

Hemos escogido esta opción ya que aun con los filtros, si el giroscopio realiza un movimiento brusco, que supere el rango de medición del MPU, se producirá un error, por lo que, al volver a la posición inicial, encontramos un cierto desfase o error.

Para esto, hemos optado por una interrupción externa mediante un pulsador.

7.6. Interrupción externa

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

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

En Arduino y Nano se dispone de dos interrupciones, 0 y 1, asociados a los pines digitales 2 y 3.

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

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

Para definir una interrupción en Arduino usamos la función:

```
attachInterrupt ( interrupt, ISR, mode );
```

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

En nuestro caso, usamos el modo LOW, ya que queremos que la interrupción se mantenga mientras tenemos el botón pulsado. Mientras esta esté activa, los leds cambiarán de color, entrará en un bucle infinito hasta que levantemos el pulsador y, al acabar, volvemos a cambiar los leds para indicar que el programa continúa funcionando correctamente antes de salir.

Además, en este caso hemos optado por inicializar los valores de los ángulos a la mitad, para eliminar cualquier error y reiniciar el contador.

```
// Interrupcion al pulsar el boton y resetea el valor del angulo

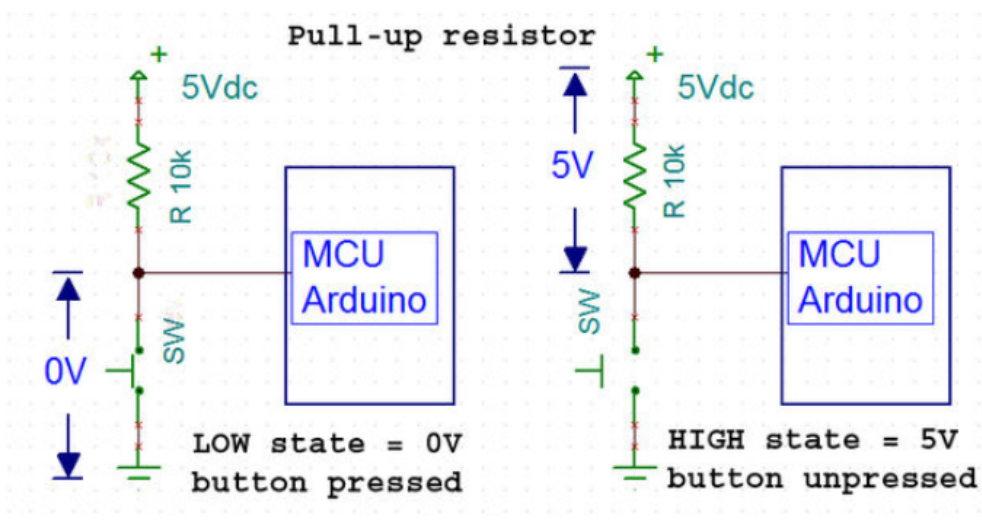
void parar()
{
    digitalWrite(Verde, LOW);
    digitalWrite(Rojo, HIGH);
    while(digitalRead(2)==LOW)
    {
    }

    Angle[0] = 90;
    Angle[1] = 90;
    Angle[2] = 90;

    digitalWrite(Rojo, LOW);
    digitalWrite(Verde, HIGH);
}
```

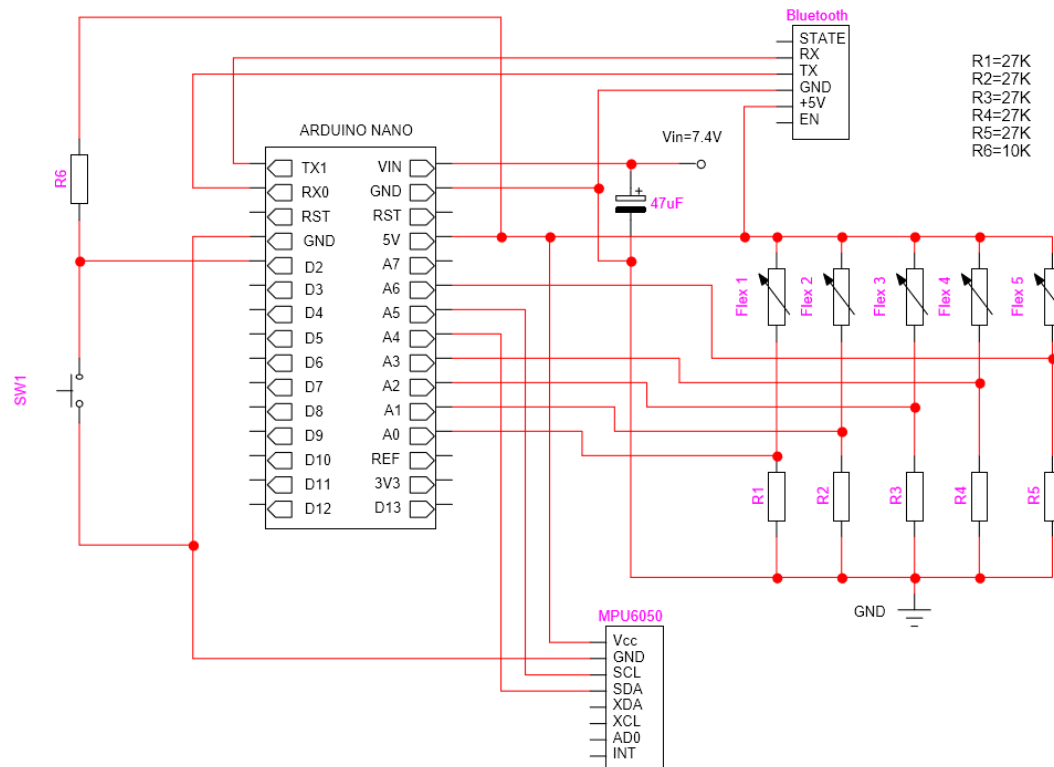
Para la conexión del botón, necesitamos una resistencia. Hay dos posibles configuraciones: Pull-up, o Pull-down.

En nuestro caso hemos optado por la opción Pull-up resistor, quedando la conexión de esta manera:



7.7. Esquemas

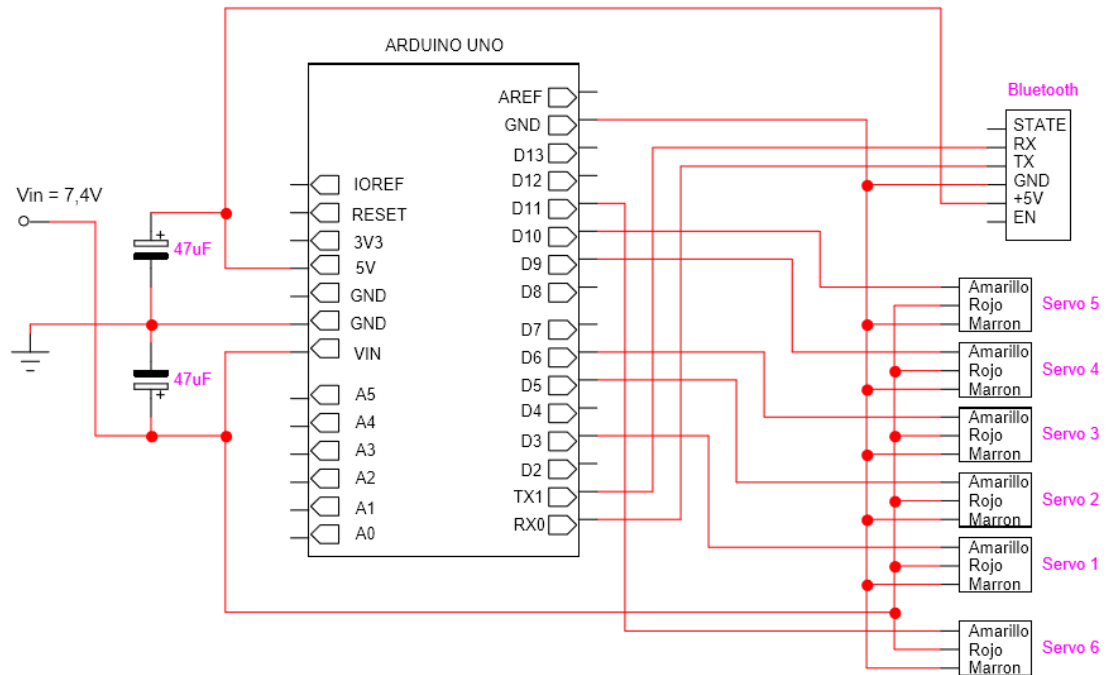
El esquema completo del guante sensor será el siguiente, en el que incorporamos el MPU6050 al Arduino Nano.



También podemos comprobar que debemos cambiar el sensor de flexión 5, el cual estaba en el pin A4, al pin A6, ya que la comunicación I2C se realiza mediante los pines A4 y A5.

Por lo que debemos también cambiar el pin mediante la programación.

Para el esquema completo del brazo robot, solamente debemos añadir el servomotor que se encargara del giro de la muñeca.



Como podemos ver en ambos esquemas, los cambios que debemos realizar a nivel electrónico son muy pocos, ya que solo debemos añadir el sensor MPU6050 y el motor que controlara el giro de la muñeca.

7.8. Código Guante

Como podemos ver, este código es el más complejo ya que es en el cual realizamos todos los cálculos relacionados con el sensor MPU6050, desde obtener los valores, hasta realizar las conversiones necesarias y enviarlos mediante bluetooth.

En primer lugar, definimos todos los tipos de datos y variables que vamos a necesitar, además del sensor MPU.

A continuación, en la función setup, inicializamos todas las comunicaciones, tanto el puerto serie como I2C, y además establecemos un patrón con los leds que nos indicaran cuando está listo Arduino.

La función principal, o main, se encargará de llamar a diferentes funciones para leer y calcular los datos del MPU, y tras esto enviarlos mediante bluetooth junto con las distintas mediciones de los demás sensores flexibles.

Por ultimo tenemos las funciones que hemos creado, las cuales nos permiten simplificar la función principal, y estas se encargan de la interrupción, leer los valores del giroscopio y calcular los valores del ángulo.

7.9. Código brazo robot

Este código es muy similar al anterior, con la diferencia de que añadimos un servomotor más, el cual movemos directamente con el valor obtenido, ya que todos los cálculos necesarios los hemos realizado en el programa del guante.

Debemos inicializar todos los motores e iniciar la comunicación serie, y tras esto identificamos mediante un carácter entre 'a' y 'f' que valor corresponde, y movemos el motor correspondiente para cada valor. Para los sensores flexibles debemos realizar un ajuste en la escala ya que esto se corresponde con la calibración de la mano. Para el mpu ésta calibración se realiza antes de ser enviada por bluetooth.

8. Bibliografía

<https://www.luisllamas.es/como-usar-un-acelerometro-arduino/>

<https://uvadoc.uva.es/bitstream/10324/12884/1/TFG-P-161.pdf>

<https://www.prometec.net/usando-el-mpu6050/>

https://naylorpmechatronics.com/blog/45_Tutorial-MPU6050-Aceler%C3%B3metro-y-Giroscopio.html

<https://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/>

<https://www.diarioelectronicohoy.com/blog/configurar-el-mpu6050>

<https://www.luisllamas.es/que-son-y-como-usar-interrupciones-en-arduino/>