**CARPETA DE CAMPO - Proyecto Informático**

**Alumno:** Mateo Basualdo

**Profesor:** Gonzalo Consorti

**Materia:** Proyecto Informático

**Curso:** 4º 1º

**Año:** 2024

**Institución:** “Escuela Técnica N” 32 D. E. 14”

**Integrantes del grupo:** Dante Portillo, Leonel Venialgo, Leonardo Stefaniszen, Mateo Basualdo y Mauricio Arzabe.

**Proyecto:** Grupo 2 - Automóvil Bluetooth

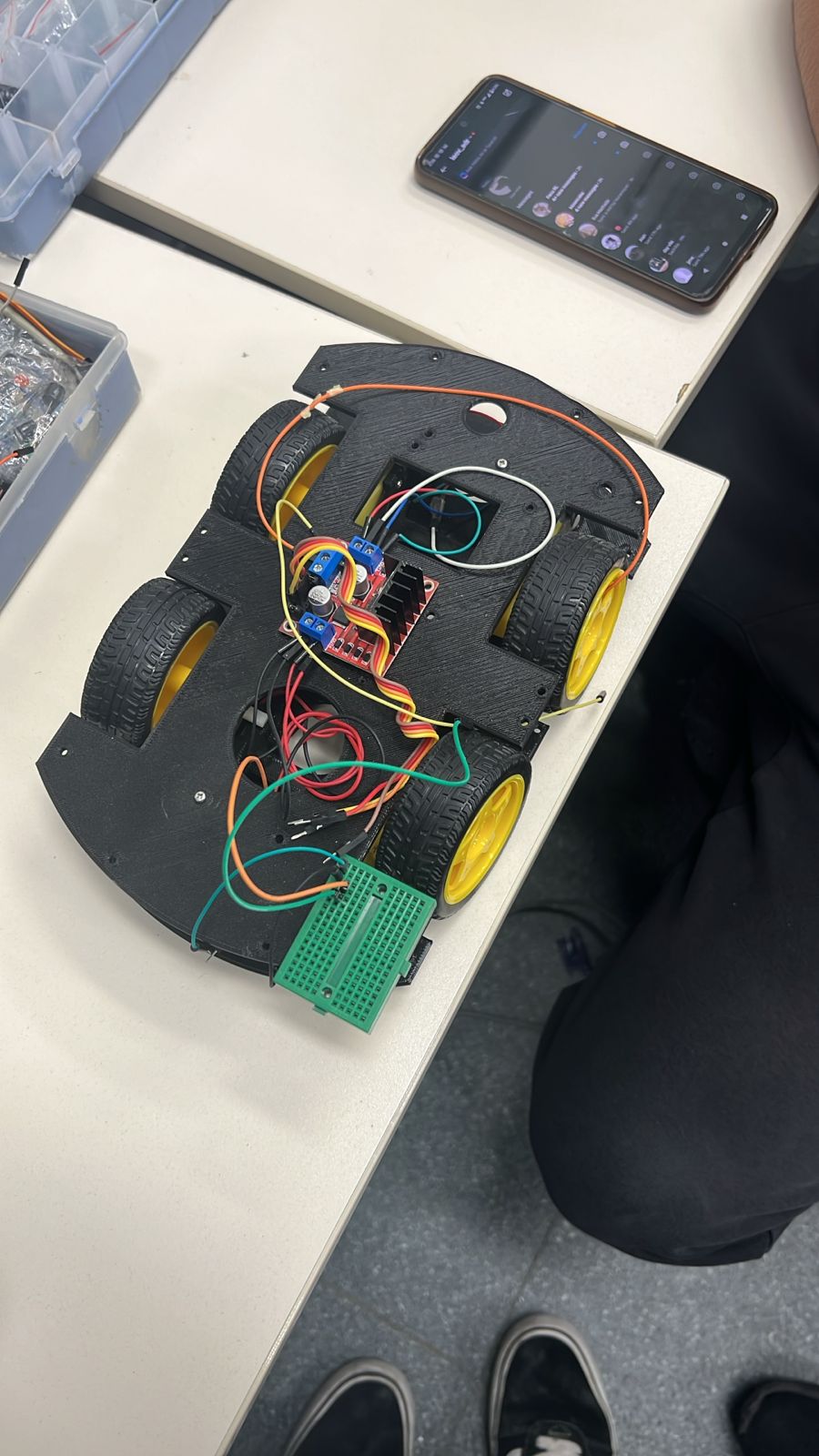


**Viernes 25/10:**

Para empezar el proyecto, primero tuvimos que definir la idea a realizar. En nuestro caso, luego de barajar distintas opciones, finalmente elegimos hacer un “Automóvil Bluetooth”. Es decir, un chasis con cuatro ruedas que se desplace en todas las direcciones (hacia adelante, reversa, giro hacia la derecha, giro hacia la izquierda, se detenga) y sea manejado mediante una aplicación del celular, el cual se conectaría al automóvil mediante una conexión Bluetooth. Si, así como los autitos a control remoto que usábamos de chiquitos, con menor estética pero más moderno.

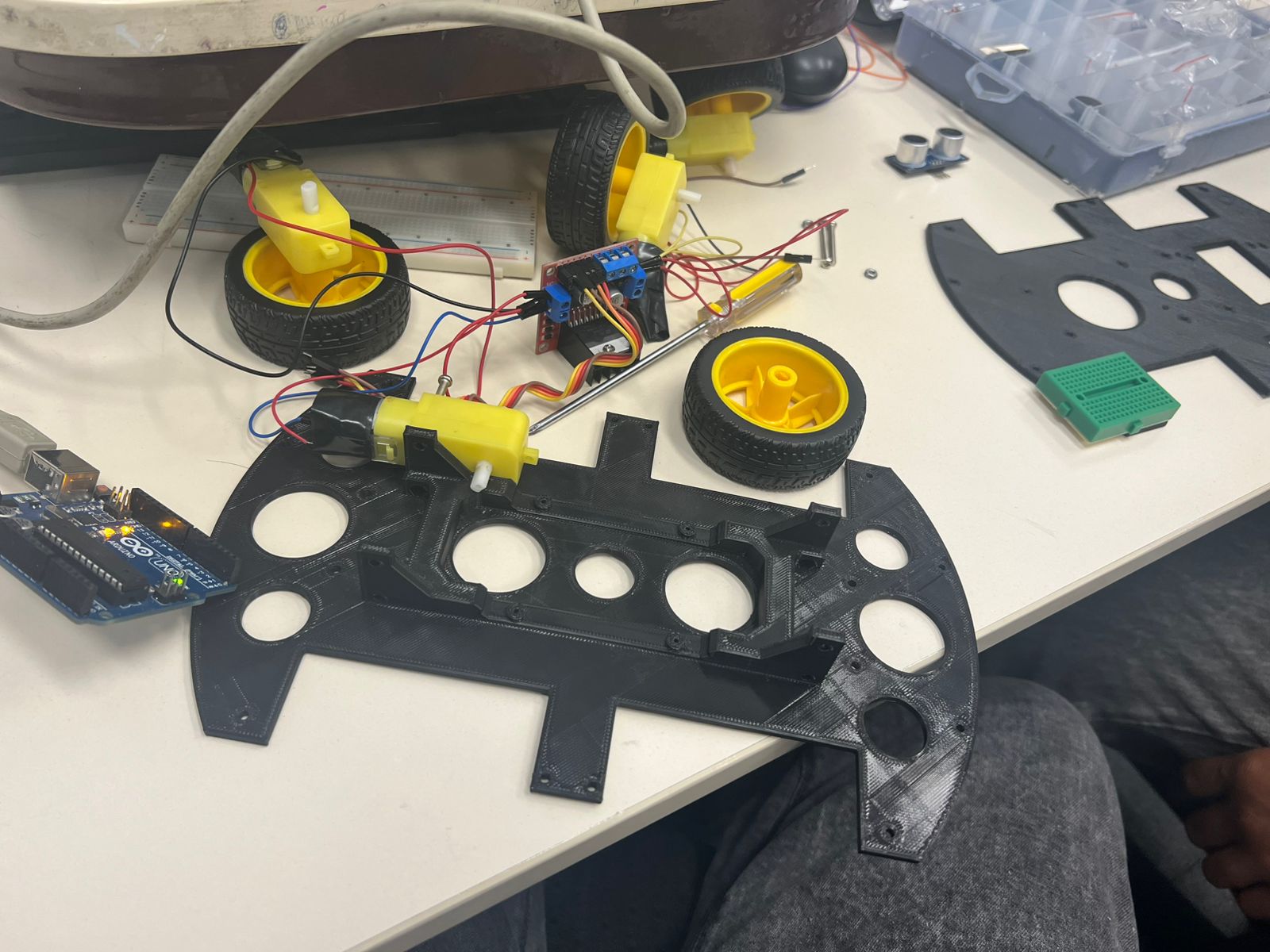
Por lo que los pilares más importantes a realizar para su funcionamiento son:

* Generar el movimiento del automóvil
* Crear la aplicación y programarla para que se pueda usar de “control remoto”
* Que la aplicación se pueda enlazar mediante Bluetooth
* Que el automóvil tenga la posibilidad recibir conexiones Bluetooth para su manejo (se me ocurre una antena o algún componente que desconozco)
* Demás funciones extras de personalización, con el objetivo de darle más aspecto.



En principio recibimos el chasis como se ve en la foto; prearmado, con las cuatro ruedas atornilladas y algunas conexiones hechas. Probablemente armado por estudiantes de otros cursos en años anteriores que quizás tuvieron la misma idea que nosotros.

Por lo que, en primer lugar, entre mis compañeros y yo desarmamos por completo las conexiones y las ruedas. Dejando descubierto el chasis para poder arrancar desde cero y a nuestra manera. Además, esto nos iba a permitir tener una mejor visión de los componentes que teníamos, para ir investigando de qué manera se utiliza cada uno y cuales íbamos a necesitar. En mi caso, desconecte una plaqueta L298N y desatornille algunas ruedas. Nos fuimos turnando con mi equipo para ir desarmando el chasis.

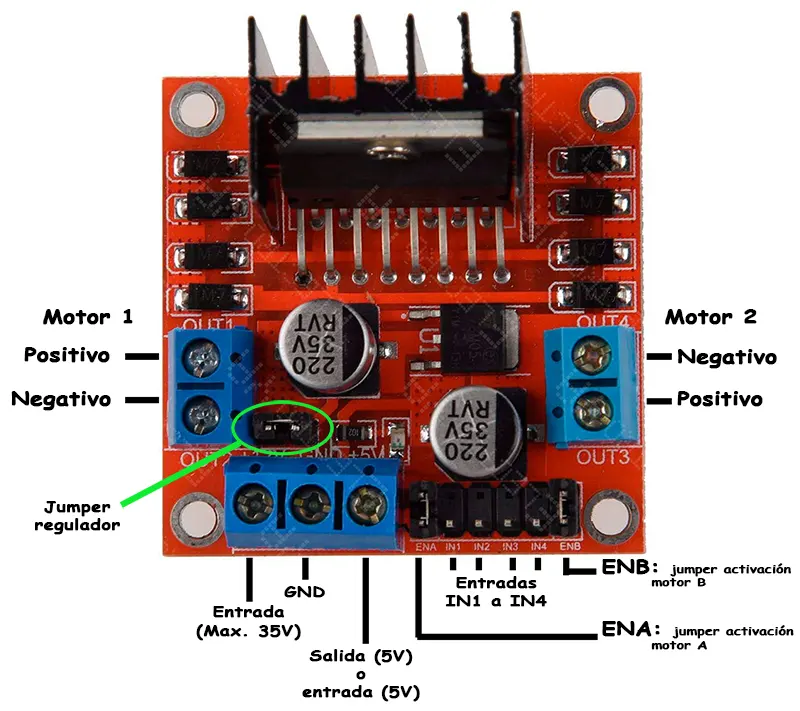


Finalmente lo desarmamos por completo y así quedó.

Una vez terminamos esto, nos permitió tener un mejor panorama de los componentes. De estos componentes me pareció curiosa la plaqueta L298N ya que no la conocía, así que me propuse con el grupo a investigar sobre esta placa para conocer su funcionamiento y analizar si nos podría servir en algún momento. En una página encontré información nueva e importante, como que esta placa era un controlador de motores, lo cual iba a ser fundamental para los cuatro motores que teníamos a disposición. Estos motores eran necesarios para que gire cada una de las ruedas y, por lo tanto, el auto se pueda desplazar. Además, es un módulo fácil de programar y permite controlar la velocidad y dirección de los motores.

Algunas características del módulo que me parecieron importantes sobre su conexión y funcionamiento fueron:

* Incorpora una entrada de voltaje
* El rango de tensiones en el que trabaja este módulo es de 3V hasta 35V
* Cuenta con una intensidad de 2A
* Posee un Jumper regulador
* Los pines IN1 e IN2 nos sirven para controlar el sentido de giro del motor 1, y los pines IN3 e IN4 el del motor 2.
* Para controlar la velocidad de giro de los motores tenemos que quitar los jumpers y usar los pines ENA y ENB.



Encontré esta imagen la cual detalla de manera completa las partes de este módulo, así que nos va a ser fundamental para el momento de la conexión.

En cuanto a su programación, la página brinda un código de ejemplo. Es sencillo, primero debemos definir todos los pines a utilizar y declararlos como salida. En el ejemplo utiliza tres pines por motor, y cada motor utiliza los siguientes puertos:

* Motor A: ENA, Entrada IN1 e IN2
* Motor B: ENB, Entrada IN3 e IN4

Se definen como salida en el código de la siguiente manera:

| pinMode (ENA, OUTPUT);  pinMode (ENB, OUTPUT);  pinMode (IN1, OUTPUT);  pinMode (IN2, OUTPUT);  pinMode (IN3, OUTPUT);  pinMode (IN4, OUTPUT); |
| --- |

Y para cada posibilidad de marcha para los motores, es decir, desplazarse hacia adelante, hacia atrás, girar para la derecha, giro hacia la izquierda y parar, tenemos que crear una función correspondiente para cada uno. Voy a dejar un ejemplo que nos brinda la pagina la cual estoy tomando como referencia, en este caso sería para girar hacia la derecha:

| void Derecha (){  //Dirección motor A  digitalWrite (IN1, HIGH);  digitalWrite (IN2, LOW);  analogWrite (ENA, 200); //Velocidad motor A  //Dirección motor B  digitalWrite (IN3, LOW);  digitalWrite (IN4, HIGH);  analogWrite (ENB, 100); //Velocidad motor A } |
| --- |

Luego para cada dirección tendríamos que hacer su función correspondiente y prender o apagar los pines según corresponda. La fuente también explica una opción para hacerlo con una biblioteca llamada ‘LEANTEC\_ControlMotor.h’. Esta opción me parece interesante pero no creo que la usemos ya que la primera opción es más sencilla de entender, por lo que no veo necesario profundizar sobre esta posibilidad en esta carpeta. Así que aquí habría terminado la primera investigación sobre este módulo controlador de motores L298N. Siento que obtuve información importante y que nos va a servir, mas cuando yo no tenía idea de para qué funcionaba. También por las dudas me gustaría volver a aclarar que los pequeños fragmentos de código que escribí anteriormente son para entender mejor y ejemplificar sobre cómo funciona y cómo programar el código. Es decir, esos códigos no son míos. La información sobre toda esta investigación fue extraída de la siguiente página:

<https://eloctavobit.com/proyectos-tutoriales-arduino/funcionamiento-del-modulo-controlador-de-motores-l298n>

Realmente me pareció una fuente muy completa sobre el tema y confiable.

**Viernes 1/11**

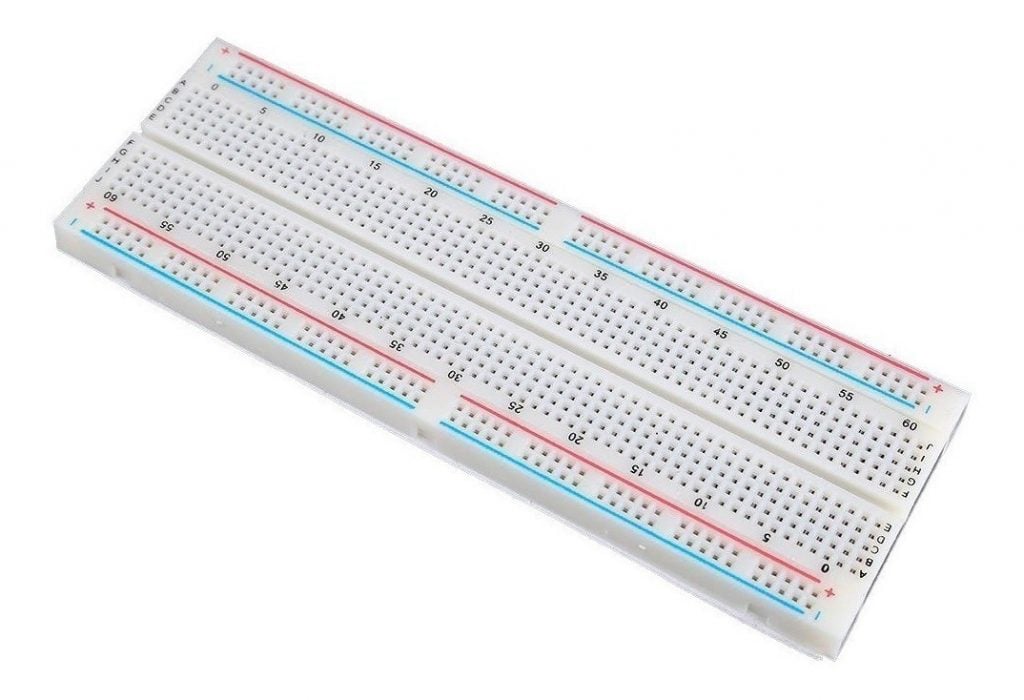
En el día de hoy, no tuvimos la materia debido a que el profe no vino, pero igualmente aproveche para seguir avanzando con el proyecto. Luego de la investigación realizada la semana anterior, me puse a separar los componentes que tenemos a disposición para mencionarlos y describir brevemente el funcionamiento y la utilidad que creo, desde mis conocimientos, que puede tener en el proyecto cada componente. Junto a unas imágenes sacadas por mi de cada uno de los componentes que disponemos, para que se pueda observar como son. Asi que arme la siguiente lista:

Componentes a utilizar (por ahora):

* Placa Arduino UNO: Es el cerebro del proyecto. Le permite al automóvil recibir información del entorno a través de sensores (como la distancia o la luz), controlar los movimientos que hace, hacer que interactúe con su entorno y comunicarlo con otros dispositivos. Es una herramienta muy versátil y fácil de usar. Dispone de pines digitales, analógicos, descarga a tierra (GND), etc. Para que se encienda y funcione debe recibir energía, para lo que debemos conectar un cable USB a una computadora. Básicamente todas las funciones que tenga el proyecto serán procesadas gracias a la placa Arduino. Hay varios modelos, en nuestro caso vamos a usar la placa Arduino UNO.



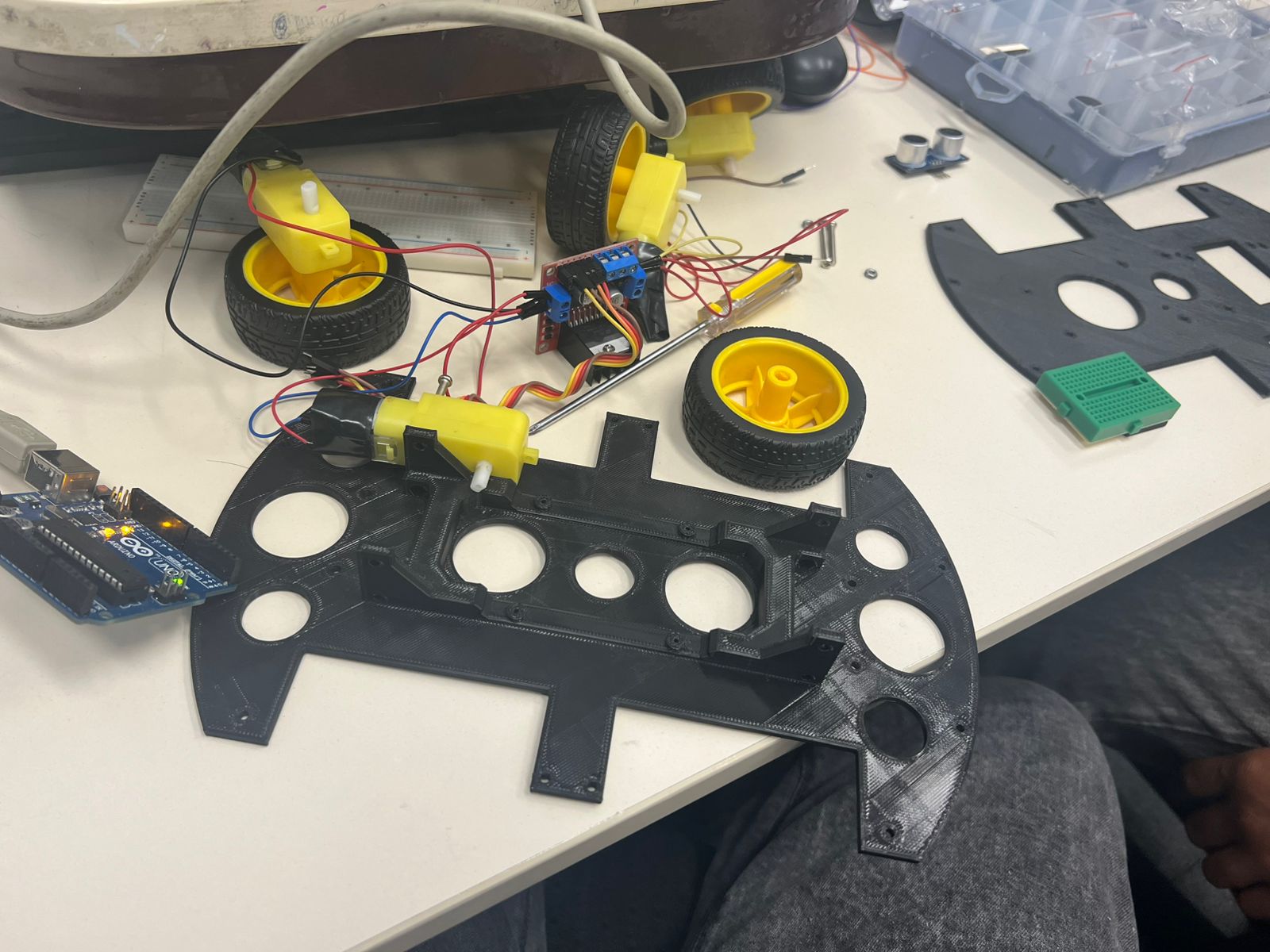
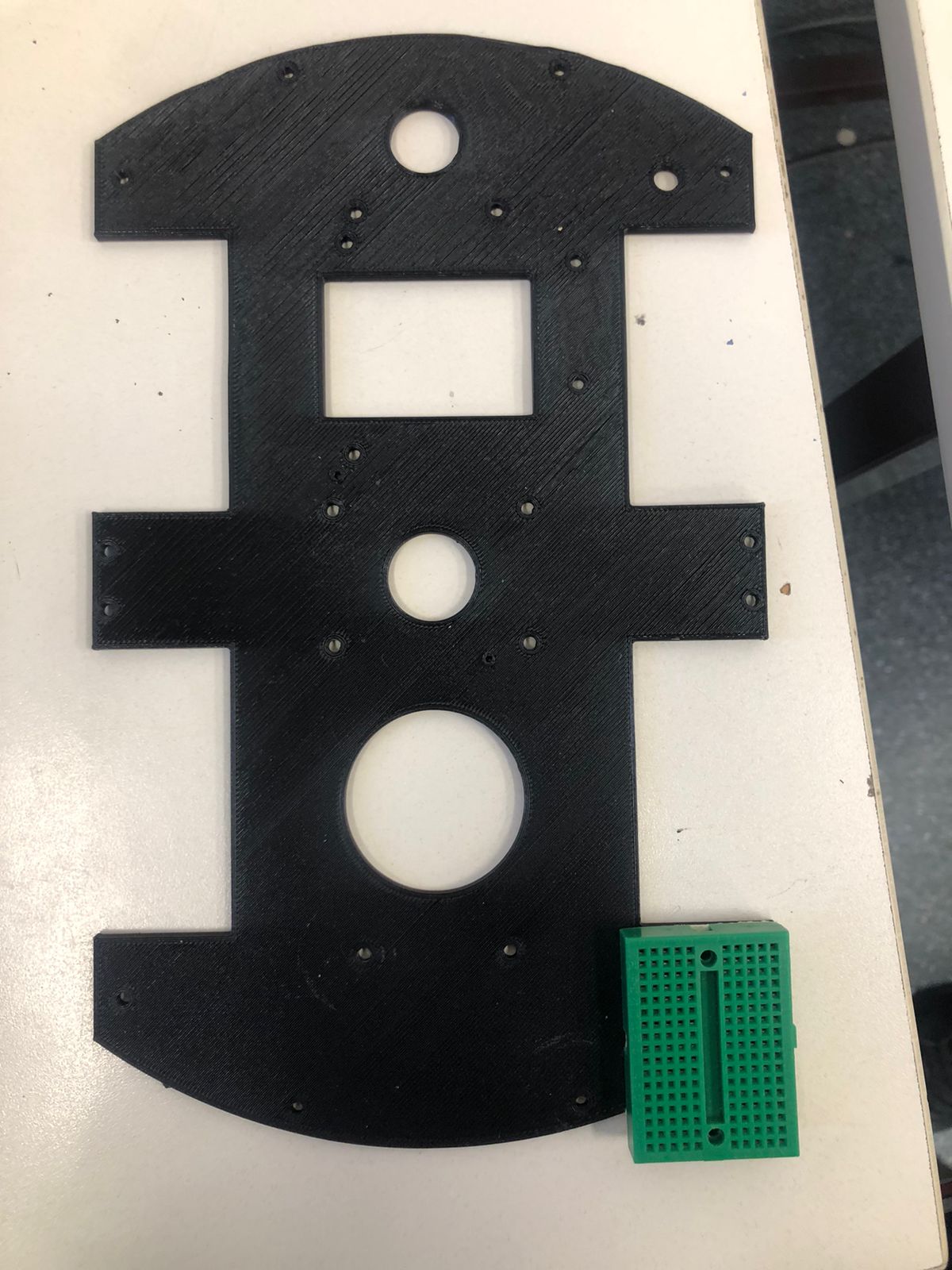
* Protoboard: La protoboard permite construir circuitos de forma rápida y sencilla sin necesidad de soldar. Es como un tablero con orificios donde se pueden conectar componentes electrónicos como resistencias, transistores, etc. Esto permite experimentar, probar diferentes configuraciones y crear prototipos para los proyectos de manera flexible y eficiente. No estoy del todo seguro todavía si nos va a servir para el proyecto final. Pero nos será útil para probar conexiones entre componentes más cómodamente, lo que requeriremos al momento de crear las funciones extras que tenga el automóvil.



* Cable USB: Su función es de permitir la conexión entre la computadora y la placa Arduino a las cuales se conecta. Es esencial para cargar el código ya que este se carga desde un programa en la computadora y se transfiere a la placa mediante este cable. Además, es el encargado de suministrarle energía a la placa Arduino para que funcione. También nos permite utilizar el Serial Monitor, siendo esto útil para depurar el código y ver ciertos valores que necesitemos. Este cable USB tiene que tener una ficha especial para conectarlo a la placa Arduino. No se el nombre de este tipo de ficha pero no es la misma que se usa para cargar los celulares, es más grande. Con esto quiero decir que no nos sirve cualquier cable USB sino que tiene que ser uno especial, con esta ficha para su conexión.



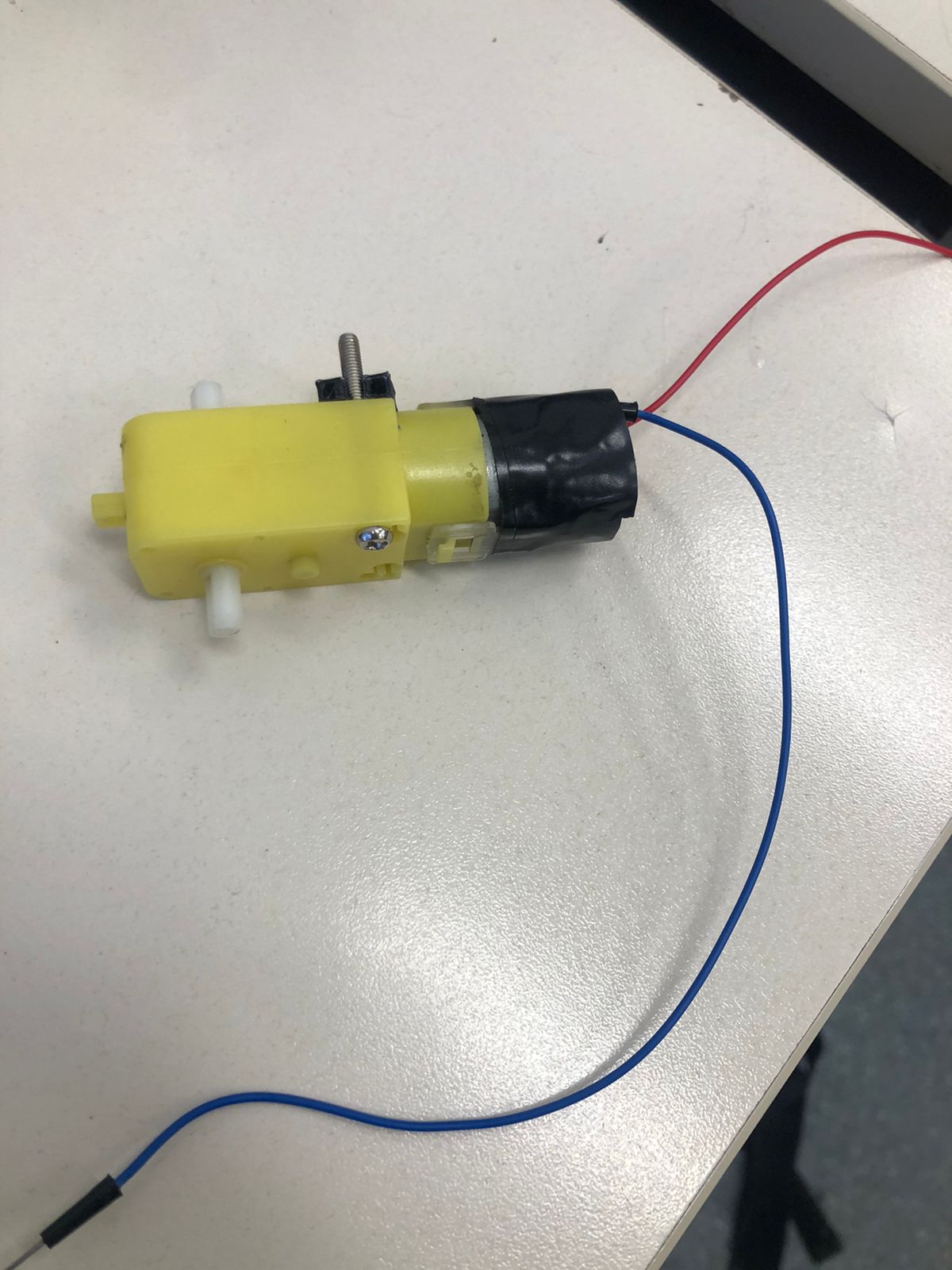
* Chasis: Es el sitio donde van a ser colocados los componentes que voy mencionando. Este no es un objeto electrónico sino que es de plástico con cierta forma para poder atornillar las ruedas y sostener los componentes. Tenemos dos partes de chasis con forma muy parecida entre ellas ya que una va arriba y la otra abajo. Como vemos, uno de ellos incluye una especie de aparato color verde el cual no se todavía para que funciona o si tiene alguna utilidad. Lo que sí recuerdo es que cuando recibimos el automóvil armado y con conexiones hechas, había unos cables conectados a este aparato, por lo que me quedará pendiente investigar para los próximos días. Este chasis por suerte ya tiene la forma necesaria para los componentes de nuestro proyecto.



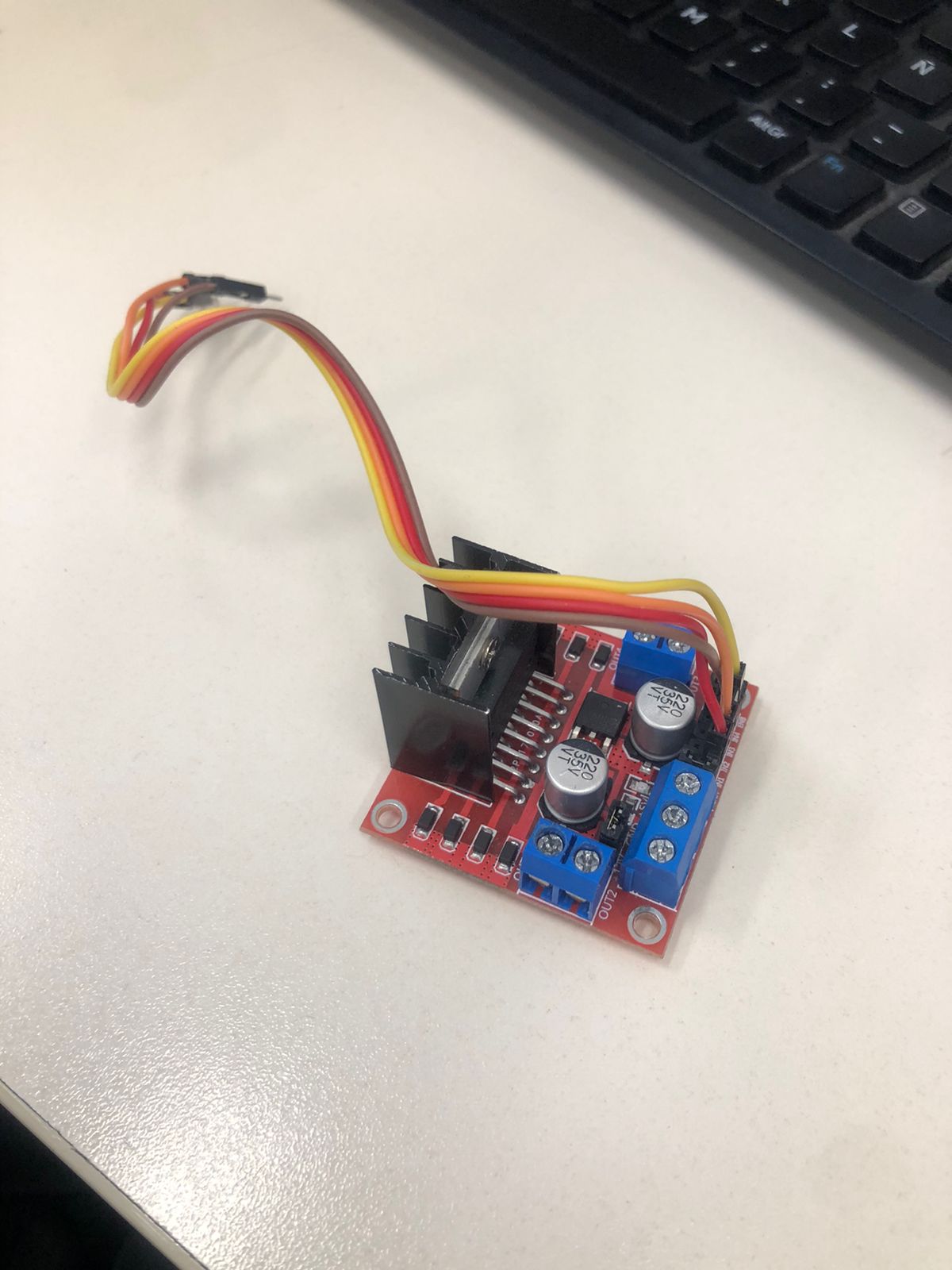
* Ruedas: Son fundamentales para nuestro proyecto ya que se encargan del movimiento del automóvil. Usaremos cuatro de estas. Trabajaran en conjunto con los motores ya que su movimiento se dará gracias a ellos. También son las encargadas de darle el giro al automóvil, en conjunto con los motores. Su único funcionamiento es rodar según la dirección en la que le indiquen los motores. Me gusta su estética.



* Motores: Son fundamentales por el hecho de que son los encargados de, como recién mencionaba, hacer que las ruedas giren. Una parte se engancha a las ruedas y la otra al chasis, por lo que usaremos cuatro (uno para cada rueda). Cada uno de estos motores vino con dos cables para conectarlos, uno a positivo y otro a negativo. Estos reciben la información desde la placa L298N, la cual investigué anteriormente, y le dan la potencia a las ruedas en la dirección programada en el código. Su correcto funcionamiento y potencia adecuada es fundamental, ya que en caso de no funcionar alguno de los motores o hacerlo con menos potencia, afectarian en gran importancia al funcionamiento del automóvil. A simple vista parece que tuvieron una gran cantidad de usos, y justamente están encintados donde se conectan los cables por algún motivo. Espero que no nos den complicaciones a futuro.



* Módulo controlador de motores L298N: El módulo controlador de motores L298N nos permite controlar la velocidad y la dirección de dos motores de corriente continua o un motor paso a paso de una forma muy sencilla. No voy a agregar mucha información más ya que lo hice en páginas anteriores cuando realice su correspondiente investigación. Lo que me deja dudas por el momento es si uno nos servirá para las cuatro ruedas o vamos a tener que utilizar dos de estos módulos. Porque por las páginas que leí hasta ahora, todas mencionan la posibilidad de controlar hasta dos motores. No se si permitirá más por lo que luego me voy a encargar de investigarlo.



* Cables de colores: nos servirán para hacer las conexiones entre los distintos dispositivos. Tenemos los comunes (macho-macho) y los macho-hembra que son útiles para hacer conexiones con componentes sin necesidad de conectarlos a la protoboard, lo que es conveniente para pequeñas conexiones.

Bibliografía utilizada para buscar ciertos puntos de los componentes mencionados:

<https://www.vobusvoice.com/es/blog/protoboard#:~:text=Protoboard%20es%20una%20herramienta%20simple,breadboard%20o%20placa%20de%20pruebas%E2%80%A6> -Protoboard

<https://www.prometec.net/l298n/#:~:text=El%20m%C3%B3dulo%20controlador%20de%20motores%20L298N%20H%2Dbridge%20nos%20permite,dos%20H%2Dbridge%20que%20monta>.

<https://www.luisllamas.es/arduino-motor-corriente-continua-l298n/>

- Controlador de motores L298N

**Viernes 8/11**

En el día de hoy, a la hora de continuar el trabajo, me encontraba bastante cansado físicamente ya que tuvimos doble turno y no había podido dormir muchas horas en la noche. También al llegar al aula, me di cuenta que faltó un integrante importante de nuestro grupo por lo que éramos uno menos. Pero eso no impidió que continúe con el proyecto, así que para empezar ocupe un rato para darle algunos retoques a lo que voy por ahora de carpeta de campo. Agregue las imágenes que utilice para la fecha anterior, fui probando distintos tipos de letras, diseños, tamaños, tipografías, etc. Hasta que un compañero me habló de las normas APA, las cuales tenía que seguir para la entrega de la carpeta. Estas normas incluyen la tipografía, tamaño y otras reglamentaciones de la letra. Por lo tanto, cambié nuevamente el documento, siguiendo esta vez estas normas.

Una vez editada la carpeta de campo según estas mencionadas normas, junto a mi compañero Dante Portillo nos propusimos probar el funcionamiento de los motores de las ruedas. Por lo que primero encastre el motor a la rueda y luego conecte la placa Arduino a la computadora mediante el cable USB. Una vez la placa Arduino ya estaba funcionando, tome el negativo de los cables del motor y lo conecte a GND, y el positivo lo conecte a 5V del Arduino. Cabe mencionar que no había distinciones entre cuál era el positivo o negativo entre los dos cables, salvo que eran de distinto color, lo cual tampoco nos decía mucho. Por lo que fue probar conectando los cables, uno a GND y otro a 5V y, si no funcionaba, invertirlos. Al conectarlos correctamente, si la rueda funcionaba empezaría a girar con cierta potencia (podía ser más fuerte o más leve dependiendo el estado del motor) y, en caso de que no funcione el motor, nos daríamos cuenta ya que la rueda no se movería ni un poco. Luego de hacer el análisis definimos lo siguiente:

* Motor 1: funciona perfectamente
* Motor 2: funciona con menos potencia
* Motor 3: funciona perfectamente
* Motor 4: NO funciona

Este resultado del breve análisis nos va a poner en problemas al momento del desplazamiento del auto, por lo que vamos a tener que buscar alguna solución. Seguramente la causa de este desperfecto de los motores sea por su largo tiempo de uso, quizás tienen ya varios años o se manipularon de forma inadecuada en el pasado, motivo por el cual se fueron deteriorando y hace que funcionen con menos potencia.

Luego de este análisis y breve informe que realice, se me ocurrió investigar lo que me había quedado pendiente la fecha anterior sobre el L298N. Lo que tenía que investigar era si el módulo controlador de motores L298N podía controlar hasta solo dos motores o más de esa cantidad, ya que nosotros tenemos que utilizar cuatro motores.

Para esta breve investigación, en vez de buscar en internet como la anterior, probé preguntándole a la IA de Google llamada “Gemini” sobre esta duda que tenía, ya que me parece que su respuesta es más directa y resalta los puntos más importantes para resolver mi duda. Elegí Gemini por sobre la otra IA conocida como “Chat GPT" porque en ocasiones me parece más completa esta primera mencionada.

Obtuve la data de que este módulo efectivamente está diseñado para controlar solo dos motores debido a su arquitectura, y a que cuenta con dos puentes H encargados de controlar cada uno a un motor. Luego profundice la investigación y le pregunté qué podía hacer si necesitaba programar más de dos motores, en mi caso cuatro. Me dio como opciones:

* Conectar otro módulo L298N: Esto implicaría una mayor complejidad en el cableado y la programación, pero no me parece mala idea. El problema es que no estoy seguro de que haya otra placa de este modelo, tendría que buscar.
* Drivers de motor más potentes: Esta opción la descarto directamente ya que por lo que tengo entendido no tenemos otro modelo de driver.
* Microcontroladores más potentes: También la descarto por el mismo tema. Me parece que no disponemos de otras placas.

Luego de esta investigación termine con más dudas que certezas, por lo que se lo voy a comentar a mis compañeros de grupo para ver que opinan y veremos cómo lo resolvemos. Por el momento tengo esa incognita, además de la de los motores que no funcionan al 100%.

**Viernes 22/11**

Hoy viernes 22 de noviembre, nos reunimos con mi grupo en la sala de informática, y definimos ciertas funciones y tareas para agregarle al auto y quien haría cada una, además de la principal que es su desplazamiento. Algunas de las funciones que se nos ocurrieron para agregarle al dispositivo fueron:

* Bocina con sonidos personalizados.
* Mini pantalla LED con una carita.
* Posibilidad del auto para doblar mientras avanza.
* Función de marcha atrás

En mi caso me decidí por hacer la bocina o claxon personalizado que iba a tener nuestro “Automóvil Bluetooth”. Con personalizada me refiero a que no será el típico sonido de bocina de los autos que se escucha y se apaga, sino una melodía que dura algunos segundos y le dé más alegría y personalización al automóvil. El tema que estaba investigando anteriormente, es decir los motores y el funcionamiento del automóvil para su desplazamiento, lo tomó otro compañero. Por lo que no me voy a enfocar tanto en ese tema por ahora, salvo que surja algún inconveniente y pueda ayudar. Igualmente en el grupo todos estamos a disposición de los demás por cualquier problema que nos surja.

Así que para hacer este claxon primero me puse a analizar qué componentes necesito utilizar para el funcionamiento de la bocina. Lo principal es obviamente el BUZZER, es el componente el cual está encargado de recibir los datos y convertirlos en sonido. Es un pequeño pero dentro de todo potente componente, y es el único del cual disponemos para emitir sonidos personalizados así que voy a usar este:



Como vemos, el buzzer tiene un pin positivo y otro negativo; el positivo se conecta al pin 5V del arduino. Y el negativo se conecta a GND (Tierra).

Los otros componentes que voy a necesitar para probar el buzzer van a ser; una placa Arduino UNO, cables de conexión y claramente el cable USB para conectar la placa Arduino a la computadora y así el Arduino pueda recibir los datos. Entonces hice la conexión simplemente para probar que funcionen los componentes correctamente. De la siguiente manera:



El buzzer comenzó a hacer un ruido bastante irritante, pero con esto comprobé que el buzzer funcionaba correctamente y estaba listo para utilizarlo. Cabe aclarar que no utilice la protoboard, ya que en ese momento estaban todas siendo utilizadas por otros compañeros. Así que me acordé de que podía conectarlos sin la necesidad de utilizar la protoboard, sino con cables macho-hembra. (La costumbre de utilizar la protoboard en el Tinkercad).

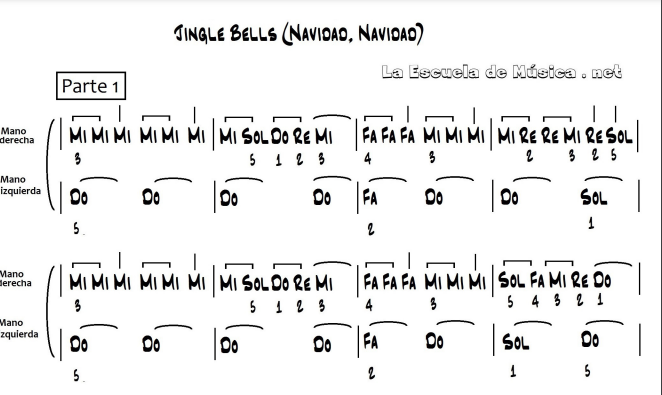
Para hacer un sonido personalizado, recordé que hace unos meses Consorti nos había explicado y había subido un TP específicamente, de cómo hacer melodías y pasarlas al Arduino, para que sean reproducidas por el buzzer. La forma para hacer las melodías era con una página llamada “Online Sequencer”:

<https://onlinesequencer.net/>

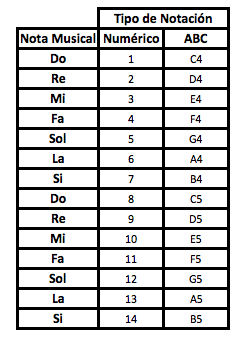
Esta página, nos permite crear melodías online sin necesidad de descargar ningún programa. Además, una vez creada la melodía la podemos exportar como archivo MIDI, y luego utilizare el siguiente transformador: <https://arduinomidi.netlify.app/> el cual transcribira el archivo MIDI al lenguaje C++ y así podemos implementarla en el arduino. Esto lo hace gracias a la función Tone:

* La función Tone() de Arduino sirve para generar un tono en un pin específico, a una frecuencia determinada y con una duración especificada en milisegundos.

Entonces me puse a probar melodías y, un compañero del grupo me preguntó si podía hacer la clásica melodía navideña de Jingle Bells, la cual le daría un toque más divertido al auto. Por lo que me puse manos a la obra con la idea de esa melodía y empecé a buscar las notas musicales, el tempo que utilizaba, el tipo de piano y demás configuraciones importantes para recrear una melodía existente. Busque en internet y utilice las siguientes imágenes para saber las notas (ya que de músico tengo poco).



Al utilizar un secuenciador online, en su interfaz las notas aparecen con letras y números, por ejemplo (C4, D4, E5, etc.). Y, debido a mi poco entendimiento sobre las notas musicales, tuve que buscar en internet también qué botón era cada nota. Encontré la siguiente imagen, la cual me sirvió bastante para saber cual era cada nota y lo fui haciendo en base a esto:



Para hacer la melodía utilice las siguientes configuraciones dentro del secuenciador:

* BPM: 130
* Instrument: Grand Piano
* Grid: 1/4
* Time signature: 4/4
* Key: C Major

Después de un rato largo probando notas, finalmente terminé la melodía. Así que la descargue como archivo MIDI y subí el archivo a la página convertidora que mencioné anteriormente. Esta página me proporcionó el código, el cual probé en la aplicación Arduino de la computadora. La primera melodía que arme y probé su código fue la siguiente:

<https://onlinesequencer.net/4313613>

Pero al probarlo en la realidad, el buzzer hacía sonidos raros y no la melodía que arme. Como si los sonidos de las distintas notas estuvieran interpuestos. En primer lugar, revisé el TP 7 en el classroom el cual trataba sobre este tema, y encontré una información la cual se me había pasado por alto y podía llegar a estar ocasionando este inconveniente. Lo que encontré fue lo siguiente:

* El Arduino **no puede reproducir dos notas simultáneamente.** Asegúrate de que tu melodía solo toque **una nota a la vez**.

¡Claro! En la melodía que arme, muchas notas se tocan en simultáneo con otras. Por lo que, no se si este era el problema por el cual no se escuchaba como debía ser, pero era una falla en el tema que debía arreglarla. Así que volví a entrar a la melodía en el secuenciador e hice las modificaciones necesarias. Una vez modificada la melodía para que esta vez suene una sola nota a la vez, repetí el proceso anterior. Descargue la nueva melodía como archivo MIDI, la subí al convertidor y este me proporcionó un código nuevamente. Conecte el buzzer al Arduino y el Arduino a la computadora mediante el USB, y subí el código nuevo. El código corre, pero nuevamente los sonidos que emite no son los correctos y se sigue escuchando de manera superpuesta. Así que tendré que investigar lo que falta. Por ahora, el enlace de la melodía a utilizar ya arreglada es el siguiente:

nueva: <https://onlinesequencer.net/4313625>

**Viernes 29/11**

Con el suceso de la semana pasada de que el buzzer no emitía correctamente el sonido, y se escuchaban sonidos interpuestos, me dispuse a investigar el por qué.

Entré a una página y empecé a leer. Encontré información nueva e importante que me podría servir para resolver el problema. Alguna información importante que encontré fue:

Tipos de buzzer:

* Pasivo: Los pasivos son aquellos que requieren tanto de la señal de voltaje como de la frecuencia de oscilación para producir el sonido. Entonces tenemos que tener en cuenta que nosotros elegimos la frecuencia de la señal para elegir el sonido que queremos que emita.
* Activo: Los activos, ya incluyen un circuito oscilador, que es un circuito que genera la señal a una frecuencia determinada. Por esto, nosotros solamente tenemos que alimentar la pequeña bocina con el voltaje adecuado y listo, el buzzer ya está listo para emitir un sonido. La única cuestión es que el sonido no lo podremos cambiar, ya está definido por el oscilador interno.

Para diferenciarlos tenemos que alimentar al buzzer con el voltaje indicado, normalmente es de 5 VCD, si emite un sonido es activo, si no emite sonido es pasivo.

Luego de leer esto en mi caso entiendo que el buzzer que utilizo es activo, ya que apenas conectarlo empieza a hacer un ruido predeterminado. También en esta página encontré un código de una melodía ya hecha el cual probé, y tampoco funciono.

La información la saqué de la siguiente página:

<https://www.automatizacionparatodos.com/musica-con-arduino-y-buzzer/>

**Domingo 1/12**

En el día de hoy, desde mi casa me puse nuevamente con el objetivo de hacer que suene correctamente la melodía que arme para el buzzer. Pensando en las posibles causas del problema que estaba teniendo, me propuse investigar y llegué a la conclusión de que me estaba faltando crear los vectores correspondientes para hacer sonar las notas. Ya que de la manera que lo estaba haciendo antes estaban las notas sueltas, con un delay muy breve, lo que probablemente estaba ocasionando el sonido erróneo. Así que como no recordaba del todo cómo funcionaban los vectores con la función Tone, entre al classroom y dentro de las clases grabadas por el profe, había una la cual explicaba este tema. Por lo que me dispuse a verlo.

Una vez termine de ver el video, abrí el Tinkercad, ya que al estar en mi casa no tenía otra manera de ir probando los conocimientos de la clase. Y en principio una de las cosas más básicas y que a la vez me estaba faltando era definir las notas.

| #define NOTE\_B0 31 #define NOTE\_C1 33 #define NOTE\_CS1 35 #define NOTE\_D1 37 #define NOTE\_DS1 39 #define NOTE\_E1 41 #define NOTE\_F1 44 #define NOTE\_FS1 46 #define NOTE\_G1 49 #define NOTE\_GS1 52 #define NOTE\_A1 55 #define NOTE\_AS1 58 #define NOTE\_B1 62 #define NOTE\_C2 65 #define SILENCIO 0 |
| --- |
|  |

Esta función lo que hace es “definir” el nombre de las notas y su frecuencia, para facilitar y saber que nota es cada una. Se explica de la siguiente manera:

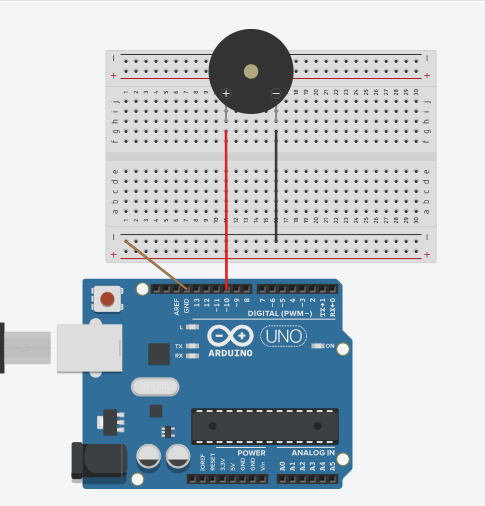
* #define: nombra a la constante según cierto nombre
* NOTE\_D2: nombre de la nota y constante
* 73: es el valor de la frecuencia asociado a la nota escrita

La lista completa de los define para todas las frecuencias y notas se encuentra en el siguiente enlace:

<https://docs.arduino.cc/built-in-examples/digital/toneMelody/>

Estas son algunas notas y es una de las cosas que me estaban faltando. La lista es más larga pero ya sería innecesario poner más en esta carpeta, igualmente en el código puse todas las notas. Tampoco es del todo necesario poner los #define de todas las notas, ya que con subir solo las notas que vamos a utilizar bastaría, pero se recomienda definir todas las notas y de allí usamos solo las que sean necesarias.

Para trabajarlo en el Tinkercad también requerimos de realizar una simple conexión; solo utilizaremos el buzzer, la placa arduino y la protoboard. Conecte el buzzer al pin 10 y la conexión fue de la siguiente manera:



Luego de un larguísimo rato probando cómo sonaba la melodía, terminé el código. Las funciones del código fueron fáciles de hacer ya que me fui guiando con el video. Sin embargo lo más costoso fue lo artístico, ya que la melodía no me convencía en cómo sonaba y estuve mucho tiempo cambiando las duraciones de las figuras, agregando silencios y demás cosas lo cual fue bastante tedioso. Pero puedo decir que me quedo bastante bien así que por ahora la voy a dejar así. Ahora, voy a pasar a ir contando el paso por paso de lo que fui haciendo en el código y como lo fui formando durante este largo rato.

**Lunes 2/12**

Para empezar vuelvo a aclarar que me guíe del video-clase de nuestro profe Gonzalo Consorti. Este video fue el siguiente:

<https://www.youtube.com/watch?v=lhP0_ssQiHQ>

Lo primero que hice fue definir al buzzer con su correspondiente pin, en este caso el 10, como se observó en la imagen previa.

#define buzzer 10

Luego, pasé a crear el primer vector, en este caso este vector es para introducir las notas musicales que lleva la melodía. Hay que colocarlas en su respectivo orden en el que van a ir sonando. Lo hice de la siguiente manera:

int melodia[] = {}

El int es para guardar datos de números enteros (como lo son las frecuencias de las notas), y melodía es el nombre que le puse para diferenciarla de otros vectores. Dentro de las llaves, van las notas de la melodía que queremos tocar, por lo que yo primero puse una breve melodía para ir probando. El vector con las notas introducidas quedo asi por el momento:

| int melodia[] = { NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_G4, NOTE\_C5, NOTE\_D5, NOTE\_E5 } |
| --- |

Las notas se introducen según su nombre por el que se definió en la larga lista anterior de las frecuencias. Y, para saber las notas que utilizaba la melodía original para introducirlas en el vector, me base en el código que me había proporcionado el convertidor de archivo MIDI a código en C++.

| tone(buzzerPin, 659);  delay(500);  noTone(buzzerPin);   tone(buzzerPin, 659);  delay(500);  noTone(buzzerPin);   tone(buzzerPin, 784);  delay(500);  noTone(buzzerPin);   tone(buzzerPin, 1047);  delay(500);  noTone(buzzerPin); |
| --- |

Esta es una pequeña parte del extenso código que me proporcionó el convertidor, pero que sirve para explicar lo que hice. Básicamente fui buscando en la tabla donde se definen las notas y reemplace dentro del vector el número que aparece después del buzzerPin por su nombre. Por ejemplo:

tone(buzzerPin, 659);  
 delay(500);  
 noTone(buzzerPin);

Si en la lista de notas buscamos la frecuencia 659 encontramos lo siguiente:

#define NOTE\_E5 659

Por lo que dentro del vector escribo NOTE\_E5 para referirme a esta nota. Y así lo fui haciendo con todas las frecuencias, una por una. Si hay una misma nota que se repite más de una vez seguida, se escribe de nuevo sin problemas. En mi caso por ejemplo tuve que ingresar en el vector tres veces seguida la misma nota ya que esta se repetía en la melodía.

Luego de reemplazar las notas y colocarlas en el vector, procedí a crear un nuevo factor de tipo int, en este caso para el tempo de la melodía, y le asigné el valor de 1000.

| int tempo = 1000; |
| --- |

Y también es necesario otro factor, esta vez de tipo float, para determinar la diferencia de tiempo entre que suena una nota y otra. El cual cree de la siguiente manera:

float factor = 1.50;

Su valor es a elección y se necesita ir probando, yo empecé con un valor de 1.30 y luego lo subí a 1.50 ya que las notas se escuchaban muy superpuestas una con la otra.

Luego debajo del primer vector creado anteriormente, forme otro vector, esta vez para que defina la duración de cada nota. La función de este vector es determinar la duración de cada nota introduciendo en él el valor de las blancas, las negras, las corcheas y las semicorcheas, según sea necesario. Estos valores dentro del arduino son los siguientes:

* Redonda = 2
* Blanca = 4
* Negra = 8
* Corchea = 16
* Semicorchea = 32

Y para los silencios:

* Silencio de Redonda = -2 con nota 0 (cero)
* Silencio de Blanca = -4 con nota 0 (cero)
* Silencio de Negra = -8con nota 0 (cero)
* Silencio de Corchea = -16con nota 0 (cero)
* Silencio de Semicorchea = -32 con nota 0 (cero)

Entonces el vector lo cree de la siguiente manera:

| int duracion[] = { 4 , 4 , 4 , 4 , 4 , 4 , 2 , 4 , 4 , 4 , 2 , 4 } |
| --- |

Y lo rellene con esos valores que, como detalle anteriormente, se puede ver que utilizo solo redondas y blancas. Las notas que tienen como duración las redondas duran un poco más de tiempo que las blancas. Cabe aclarar que este vector funciona en paralelo con el otro vector de las frecuencias, es decir que a la primera frecuencia del otro vector le corresponde la primera duración de este vector llamado ‘duración’, o sea una blanca. La segunda frecuencia tiene como duración el segundo dato de este vector, nuevamente una blanca, y así sucesivamente. Esto me generó bastantes dolores de cabeza ya que continuamente se me desordenaba algún vector y me perdía de que frecuencia iba con cuál duración. Así que utilice un consejo del profe Gonzalo Consorti que menciono en la video-clase que me sirvió bastante fue de ordenar un vector abajo del otro y tener cada dato ordenado con su correspondiente, de la siguiente manera:

int melodia[] = { NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5, NOTE\_E5 };  
  
int duracion[] = { 4 , 4 , 4 , 4 , 4 };

Obviamente está acortado, pero de esta forma iban quedando ambos vectores, el de las notas y el de las duraciones. Como comente, los ordene de esta forma para que como se ve, sea más fácil de saber que duración le corresponde a cada nota. No me hizo falta saber mucho de teoria musical para elegir la duración de cada nota, ya que era un poco intuitivo según como escuchaba que sonaba, pero a la vez bastante duradero y agobiante, ya que tenía que reproducir la melodía, hacer las modificaciones necesarias y volver a reproducir la melodía para ver como sonaba, y así repetidamente para cada cambio. Me parece importante aclarar también que en las últimas dos líneas de código que subí a esta carpeta mostrando cómo habían quedado los dos vectores, en el vector llamado ‘melodia’ son todas las mismas notas ya que el comienzo de la melodía es así. Es decir no se trata de ningún error, si seguimos con el código se pueden observar las otras frecuencias que utilice para la melodía.

Una vez ya tenía las frecuencias o notas ingresadas en su vector y su cierta duración ingresada en el otro vector, faltaba crear una función para indicarle al arduino que las reproduzca mediante el buzzer conectado. Para esto cree un ciclo for y dentro de él, las demás instrucciones para ejecutar por ese ciclo, las cuales voy a ir repasando más adelante.

| for(int i = 0 ; i < sizeof (melodia) /2 ; i++ )    {  tone (buzzer, melodia[ i ] );  } |
| --- |

Este ciclo ‘for’ lo que hace es que se reproduzca todo el vector llamado ‘melodia’, es decir el de las notas. Dentro del ciclo ‘for’ llame a la función tone mencionada anteriormente, y le indique que el pin llamado ‘buzzer’, reproduzca el vector llamado ‘melodia’. También dividi por dos el sizeof del vector para evitar problemas y que se ejecute correctamente. Pero esto no termina acá, ya que tuve que agregar una variable dentro del ‘for’ para definir cuánto va a durar cada nota. Básicamente le tengo que dar la funcionalidad al vector creado anteriormente llamado ‘duracion’, ya que si tiene los números sueltos, el arduino no entiende si es una blanca, una negra, una corchea, una semicorchea o que es. Esto lo hice de la siguiente manera:

int duracionNota = tempo / duracion[ i ];

Esta variable la cual llame ‘duracionNota’ es la encargada de darle la duración a cada nota como su nombre bien lo dice. Su funcionamiento es sencillo y se basa en que divide el tempo establecido anteriormente sobre la duración de cada nota (el vector ‘duración’). Por ello, si en la duración de cada nota elijo una corchea la cual vale 16 cuadraditos en el arduino, la duración de esa nota será menor ya que esta función dividiría el tempo por dieciséis. Para una nota más larga, tendría que usar la redonda la cual vale dos cuadraditos, por lo que al dividirlo, le daría un número más grande a la variable ‘duracionNota’ y la nota va a ser más larga. Otra opción para que las notas sean más largas sería subir el tempo, ya que con la misma lógica que explique recientemente, al subir el tempo la división de la variable ‘duracionNota’ va a dar un resultado más grande por lo que la nota durará un pequeño tiempo más. Así que una vez creada esta variable la agregue a la función tone y juntando todo el código por el momento quedo asi:

| for(int i = 0 ; i < sizeof (melodia) /2 ; i++ )    {    int duracionNota = tempo / duracion[ i ];    tone (buzzer, melodia[ i ], duracionNota );  } |
| --- |

Parecía terminado, sin embargo, al ejecutarlo se escuchaba mal, con las notas muy rápidas y superpuestas unas con las otras que hasta parecía solo un sonido raro y estaba muy lejos de ser una melodía navideña con notas musicales. Al principio no entendía por qué estaba pasando esto, por lo que seguí viendo la clase y entendí que era porque hacía falta una pausa. Es necesaria una pausa ya que el arduino ejecuta las acciones una atrás de la otra en cuestión de microsegundos, por lo que la melodía suena pero como las notas se ejecutan en microsegundos no se alcanzan a diferenciar y por eso se termina escuchando un solo sonido raro. Para hacer esta pausa fue necesario crear una nueva variable la cual llame ‘pausa’:

| int pausa = duracionNota \* factor;  delay( pausa ); |
| --- |

Lo que hago con este código es; primero crear una variable de tipo int, la cual yo la llame pausa. El valor de esta variable es la multiplicación de la variable llamada ‘duracionNota’ (creada anteriormente a esta) por el factor, el cual fue una de las primeras cosas que hice. Recordando, el factor que yo puse tiene un valor 1.50, pero se puede modificar a gusto. Y luego simplemente agregue el delay, es decir la pausa que va a tener, con el correspondiente valor de esta variable creada llamada ‘pausa’. Así cree la pausa necesaria entre las notas para que se distingan una de las otras y no suenen tan rápido que no se alcancen a percibir las distintas frecuencias como pasaba anteriormente.

Finalmente me faltó poner la última instrucción en el código, la cual era la función llamada ‘noTone’. Esta función utilizada para el buzzer lo que hace es lo opuesto al Tone. Es decir, le indica a la placa arduino que no mande ningún sonido al buzzer, por lo que el buzzer no emite ninguna frecuencia. Así que por último agregue en el código, debajo de las anteriores instrucciones, esta función mencionada:

noTone( buzzer );

Así que terminado esto, la función de void setup( ) quedó armada de la siguiente manera:

| void setup() {  pinMode(buzzer, OUTPUT);    for(int i = 0 ; i < sizeof (melodia) /2 ; i++ )  {  int duracionNota = tempo / duracion[ i ];    tone (buzzer, melodia[ i ], duracionNota );    int pausa = duracionNota \* factor;  delay( pausa );  noTone( buzzer );   } } |
| --- |

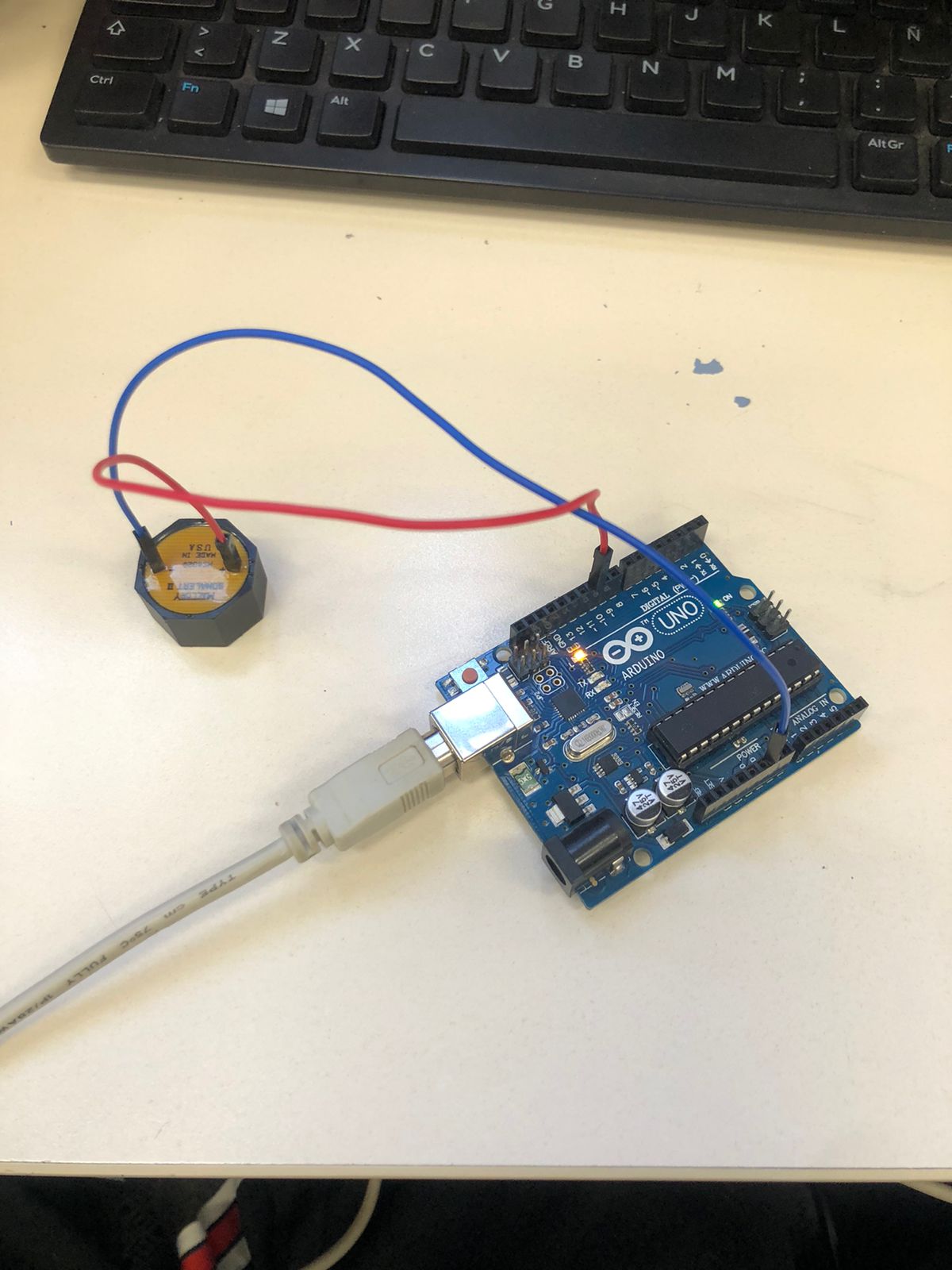
Y la parte del void loop( ) la deje vacía ya que la idea de la melodía es que empiece y una vez finalice no se vuelva a repetir, por eso el código está escrito en el void setup( ).

Así que una vez ya tenía terminada la parte importante que es el funcionamiento de la melodía, al probarla sonaba bien, pero no del todo como me gustaría. Sentía que le faltaba una separación entre algunas notas, por lo que para esto me iba a servir el pin de frecuencia 0, definido anteriormente como SILENCIO. Esta nota, como bien lo dice su nombre, deja un silencio como una nota más, ya que su frecuencia es de 0. Lo cual sirve para las canciones ya que en la mayoría de casos no suenan todas las notas seguidas, sino que hay una breve separación en cierto momento. Entonces me puse a agregar los silencios correspondientes en cada momento según la melodía original. Esto me tomó un largo rato ya que cada silencio que agregaba tenía que probar la melodía para ver como iba quedando, y muchas veces no quedaba como esperaba así que fue un rato de prueba y error en la melodía y de agregar y borrar notas o silencios. También cambie alguna que otra duración de las notas cuando sentía que debían ser más largas o, en su defecto, más cortas.

**Miércoles 4/12**

En el día de hoy, en el horario de matemática el cual tuvimos libre ya que no teníamos que rendir nada, nos juntamos con el grupo para cada uno continuar con su parte y arreglar los problemas que tenía el automóvil. El problema era que los motores dejaron de funcionar y, por lo tanto, el auto no arranca. Hace ya varios días que surgio este inconveniente, y no sabemos exactamente cuál es la falla. Probablemente sea que los motores se gastaron demasiado, que el módulo controlador L298N no funcione o alguna otra falla interna la cual sabemos.

Un rato después de escribir el párrafo anterior, junto a mi compañero Leo Stefaniszen estuvimos probando el automóvil y los motores, probamos muchos tipos de conexiones y nuevamente no funcionaban. Hasta el módulo controlador ni prendia las luces por lo que supusimos que el problema era este módulo, el cual directamente no funcionaba. Así que fuimos al aula donde se encontraba Consorti el día de hoy para ver si él encontraba la solución. Consorti desconecto y volvio a conectar los cables y empezó a funcionar el módulo y con él, dos motores. Por lo que había dos motores funcionando y dos que no, pero teníamos la noticia positiva de que el módulo controlador si que servía. Nos pusimos a probar cada motor por separado, y definimos que había dos motores que funcionaban correctamente, otro que arrancaba muy despacio y el otro directamente no funcionaba. Hablé con mi compañero y definimos que él se quede con Consorti viendo si encontraban la solución y yo volvería a nuestro aula a continuar con mi parte del proyecto.

Por lo que volví al aula a seguir con el tema de la melodía. El código ya lo había hecho ayer en Tinkercad así que ahora solo tenía que probarlo en la realidad, para que se escuche correctamente. Únicamente agregue los define correspondientes solo para las notas que utilice, ya que la lista de todas las notas no es necesaria escribirla entera, sino que solo las que voy a utilizar. También defini el pin 10 en el cual conecte el buzzer. Así que hice la conexión básica con el buzzer, la placa Arduino y los cables. Luego subí el código a la aplicación de Arduino en la computadora y lo ejecuté. 

| #define NOTE\_E5 659 #define NOTE\_G4 392 #define NOTE\_C5 523 #define NOTE\_D5 587 #define NOTE\_F5 698 #define NOTE\_G5 784 #define NOTE\_C6 1047 #define NOTE\_D6 1175 #define SILENCIO 0  #define buzzer 10 |
| --- |

Al ejecutarlo se escuchaba de manera correcta como lo había hecho en el Tinkercad, por lo que fue un éxito y ya termine con esta melodía.

Sin embargo la idea del grupo era hacer mínimamente dos melodías, así que como no tenía otra tarea para hacer en el proyecto ya que todas están cubiertas por mis otros compañeros, abrí el secuenciador y me puse a crear otra melodía desde cero. Esta vez voy a ir explicando un poco más por encima lo que fui haciendo ya que es muy parecido a lo que ya explique, básicamente porque se trata de lo mismo, solo que con otras notas y configuraciones. Mientras pensaba que melodía crear, un compañero me dio la idea de hacer la famosa melodía de la canción “Crazy Frog”. Por si no se conoce la canción, es la siguiente:

<https://www.youtube.com/watch?v=k85mRPqvMbE>

Así que me pareció una buena melodía para hacer y me puse a trabajar en el secuenciador online. Luego de un rato termine la melodía. Al ser ya la segunda melodía que armaba, ya sabía el camino que tenía que seguir y no cometí los mismos errores que en la primera oportunidad, ya que como dice el dicho, de los errores se aprende. Por ejemplo, ya sabía que no se podían interponer unas notas con las otras ya que el arduino puede hacer sonar solo una nota a la vez Esto me permitió avanzar más rápido y, al terminarla, la descargue como archivo MIDI y en el convertidor la pase a código C++. La melodía final fue la siguiente:

<https://onlinesequencer.net/4335806>

Con el código que me proporcionó lo que fui haciendo es ir viendo qué notas utilice, y las iba definiendo con el define, para tener definidas solo las notas a utilizar. Una vez fui una por una chequeando las notas y pasándolas al define, me quedo de esta manera:

| #define NOTE\_B3 247 #define NOTE\_D4 294 #define NOTE\_E4 330 #define NOTE\_F4 349 #define NOTE\_FS4 370 #define NOTE\_G4 392 #define NOTE\_A4 440 #define NOTE\_B4 494 #define NOTE\_C5 523 #define NOTE\_E5 659 #define SILENCIO 0 #define buzzer 10 |
| --- |
|  |

Estas fueron las notas que supe que iba a utilizar, lo que me permitió estar más organizado y ahorrar muchas líneas de código innecesarias.

Luego copie y pegue el mismo ciclo for y las funciones que contenía en el void setup( ), ya que básicamente era lo mismo. Y ahora me tocaba la parte más tediosa, colocar las notas y sus duraciones en los vectores y lugares correspondientes. Los nombres de los vectores los deje iguales a la anterior melodía y comencé a pasar las notas al vector llamado “melodia”. Me iba basando en el código que me proporcionó el convertidor, y fui poniendo una por una hasta llegar hasta la mitad de la melodía, ya que, en este momento la melodía se repite de la misma forma que en la primera mitad. Por lo que fue simplemente copiar y pegar en la segunda mitad para ahorrar tiempo. Una vez tenía las notas cargadas, tenía que determinar la duración de cada una, así que fui una por una escribiendo la duración según lo que me parecía al principio. Lo que sabía es que esta melodía era más rápida que la anterior, por lo que en primer lugar le subí bastante el tempo a esta melodía. Recordemos que el tempo de la anterior era de 1000 BPM debido a que su ritmo era lento.

| int tempo = 1400; float factor = 1.50; |
| --- |

Use esas configuraciones de tempo y factor. Otra característica que tenía esta canción al ser más rápida, es que tenía que usar duraciones más cortas para las notas de las que había usado en la anterior. En este caso empecé probando mayormente las corcheas que valen 16, en la mayoría de las notas y algunas negras de valor 8 en las restantes. Sin embargo al probar el código la melodía se reproducio excesivamente rápido, así que baje el valor a 8 (negras) de todas las notas. Al hacer este cambio iba tomando forma la melodía pero le faltaban retoques, alargue algunas notas cuando era necesario, agregue varias pausas (SILENCIO) y añadi alguna que otra nota mas cuando me pareció necesario para darle más fluidez y sentido a la melodía. Finalmente termine la melodía. Tomó su tiempo pero fue mucho más rápido que la anterior y quedé conforme con el resultado. Ya terminado se me ocurrió hacerle una breve modificación; a diferencia de la anterior melodía que sonaba una vez y se apagaba debido a que el código se encontraba en el void setup(), en esta quise experimentar de realizarlo en el void loop( ), lo que haría que una vez finalice la melodía vuelva a empezar y así infinitamente hasta que se corte la conexión. Así que el mismo código que tenía armado en el void setup( ) lo pegue en el void loop( ), a excepción sólo del pinMode del buzzer. A esto únicamente le tuve que agregar un breve delay final para indicarle al arduino cuánto tiempo tenía que esperar para repetir la melodía. Finalmente el void loop() quedó armado de la siguiente manera:

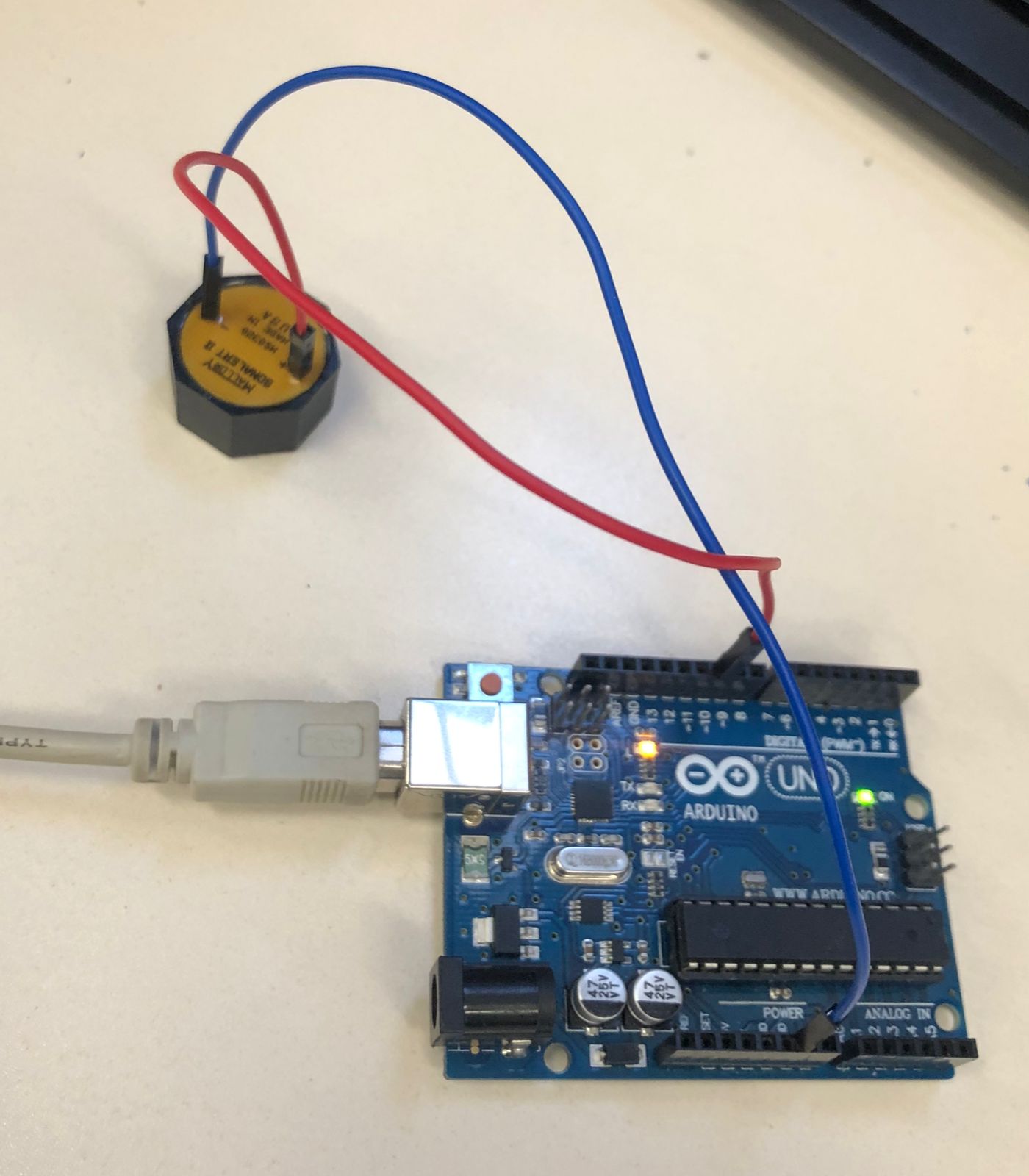
| void loop() {  for(int i = 0 ; i < sizeof (melodia) /2 ; i++ )    {  int duracionNota = tempo / duracion[ i ];    tone (buzzer, melodia[ i ], duracionNota );    int pausa = duracionNota \* factor;  delay( pausa );    noTone( buzzer );  }  delay(400); } |
| --- |

Lo probé y funcionó perfectamente, así que ya tenemos dos melodías para el claxon del automóvil. Esta segunda melodía ya la hice desde mi casa mediante el Tinkercad por lo que me faltaría probarlo con los componentes físicos, pero todo parece indicar que se escucha correctamente.

Ya no nos quedan cosas para hacer en el proyecto por lo que sería probar mañana la melodía en el buzzer físico y ya me despediría de esta carpeta.

**Jueves 5/12**

En el día de hoy, asistí al colegio con el último objetivo de probar la segunda melodía que arme ayer. Así que en el momento libre que tuvimos en base de datos ya que la profesora no asistió, hice la conexión correspondiente del buzzer en la placa arduino. Copie el código desde Tinkercad y lo pegue en la aplicación de Arduino en la computadora. Al ejecutarlo, se escuchaba tal y como lo esperaba, y se repetía infinitamente hasta que cortaba la conexión.



Por lo que mi trabajo estaba finalizado, solo queda terminar de organizar con mis compañeros como vamos a hacer la exposición del día de mañana. Lamentablemente el automóvil no parece querer funcionar para mañana, debido a que de los cuatro motores necesarios para su funcionamiento, solo arrancan dos. Igualmente vamos a presentar todas las funcionalidades que hicimos en este, aproximadamente, mes y medio. Gracias por leer estas 37 largas páginas.