**Arduino: Funciones - BUS Serial - Sensores de Ultrasonido**

*Álvarez, Mariana.*

*Sellanes, Paula.*

*Grupo B3*

*14/4/2020*

**Link del proyecto:** https://www.tinkercad.com/things/76KpqVCMhQT-ingenious-vihelmo/editel?sharecode=YCdkBsikzxccoZdFR1Ng21j\_6kQ9q5DKG1iNJPw1IeI

1. **Resumen**
2. **Objetivo:** Utilizando Tinkercad como herramienta, agregar al circuito previo (de las clases pasadas) de los motores CC, dos sensores ultrasónicos en paralelo. A partir de esta base desarrollar posteriormente un código que sea capaz de detectar un objeto a una distancia (a través estos sensores) para guiar al robot hacia el mismo.
3. **Resultados:** Guiar al robot hacia un objeto detectado por los sensores ultrasónicos.
4. **Introducción**

Acompañado con el desarrollo del curso esta actividad complementa lo trabajado previamente, con la clase cinco. Contando con las bases de puente H, la variación en el sentido de los giros en motores de corriente continua y todas las conexiones que estos implican, le agregamos a este trabajo la comunicación serial y los sensores ultrasónicos.

El objetivo radica en poder fusionar el sentido de giro de los motores con la dirección a tomar (derecha, izquierda, adelante, atrás o detenido) y finalmente hacer que estas dependan de la posición relativa de un objeto externo.

Como método de contextualización con lo que trabajaremos, definimos funciones como un conjunto de declaraciones, definiciones, expresiones y sentencias que realizan una tarea específica. Estas conforman un código que cumple una tarea o múltiples tareas determinadas. Las creamos cuando se necesita realizar la misma acción múltiples veces dentro de un mismo programa. Un ejemplo son las funciones esenciales de Arduino son *setup() y loop()* son las más utilizadas y vienen por defecto.

Otro término a definir es el alcance de una variable, esta es una propiedad de programación que utiliza Arduino, dependiendo donde se definan estas va a ser su alcance. A las mismas las podemos clasificar en locales y globales:

1. Las variables locales se definen dentro de una función y solo se pueden ser utilizadas allí. Son útiles para determinar que una función tiene acceso a sus propias variables, evitando así errores de programación cuando se modifica la variable repentinamente utilizadas por otra función.
2. Por otro lado una variable global está definida en el programa principal (fuera de cualquier función que sea implementada), es decir, que puede ser usada en todo el programa luego de declararse. Puede ser modificada dentro de la implementación pero no es recomendable.

Por otro lado encontramos a los motores de corriente continua (previamente mencionados), los cuales tienen únicamente dos conductores (uno de ellos es positivo y el otro negativo). Si estos dos conductores son conectados directamente a la batería el motor gira, y si se realiza el intercambio de conexión de estos el motor seguirá girando pero en dirección opuesta. Es importante recordar que no se debe accionar el motor directamente desde los pines del la placa del Arduino, pues puede dañar el microcontrolador. Para su funcionamiento se debe usar un circuito de controladores o un circuito integrado.

Para controlar el giro de dirección de un motor de corriente continua (sin intercambiar los conectores) utilizaremos un circuito llamado *H-Bridge*. Un puente en H es un circuito electrónico que puede accionar el motor en ambas direcciones. Estos son utilizados en varias aplicaciones diferentes, la más común de estas es el control de motores en robots. Se le llama puente H ya que utiliza cuatro transistores conectados de forma similar que hace que el diagrama esquemático se parezca a una “H”.

En el resto del informe se encontrarán contenidos que respaldaron la práctica (o eso tratamos que hagan)

1. **Descripción de la experiencia**
2. **Materiales:**

* Arduino UNO
* 2 motores CC
* Cables
* 2 sensores ultrasónicos
* Batería 9V
* Controlador de motor puente H
* Placa de pruebas
* Tinkercad

1. **Procedimiento general:**
2. Armado del circuito de dos motores cc[[1]](#footnote-0) conectados a través de un puerto H[[2]](#footnote-1) a un Arduino UNO.
3. Creación de un código que pueda a través de la comunicación serial con el Arduino enviar ciertos patrones de “conducta” (en otras palabras direcciones a tomar) a través de valores de presentes en la tabla ascii.
4. Incluir en el circuito dos sensores ultrasónicos en paralelo.
5. Crear un código tal que nuestro robot avance en la dirección de un objeto sensado por los sensores.
6. **Implementación de la comunicación serial:**

La palabra serial en sí misma significa “uno después de otro” y en realidad establece las bases del concepto, la comunicación serial se da cuando transferimos datos un bit a la vez, uno después de otro. La información se transfiere entre la computadora y el Arduino, esencialmente, estableciendo un pin *HIGH* o *LOW* (es una técnica parecida para prender una *LED* pero son datos). Esta comunicación ocurre utilizando *RX (pin 0)* y *TX (pin 1)* del Arduino. En otras palabras, un lado de la comunicación establece el pin y el otro lo lee.

Dentro de la comunicación serial encontramos (luego a usar):

* ***“Serial.begin(9600)”***, este procedimiento se establece en el *setup()* del código y lo que hace es establecer la comunicación serial a 9600 bits por segundo.
* ***“Serial.println()”*** , esto significa que llama a un procedimiento llamado *println* que significa imprimir línea. Lo que hace es enviar datos desde el Arduino a la computadora.
* ***“Serial.available()”*** , se utiliza para saber si hay datos en el buffer[[3]](#footnote-2) que leer, si no hay datos envía un cero.
* **“Serial.read()”**, se encarga de leer los datos del buffer.

1. **Implementación de los sensores:**

El sensor ultrasónico Arduino nos permite medir distancias a través de los ultrasonidos. Este utiliza las propiedades de propagación del sonido para medir distancias, en concreto utiliza los ultrasonidos.

El sensor envía una onda ultrasónica a través del ***“Trigger”***, rebota contra el objeto y el ***“Echo”*** detecta la onda, esto sucede ya que el sensor dispone de un micrófono adecuado para esa frecuencia.[[4]](#footnote-3) Midiendo el tiempo entre pulsos, conociendo la velocidad del sonido, podemos estimar la distancia del objeto contra cuya superficie impactó la onda/pulso de los ultrasonidos.

Sabemos que la velocidad del sonido es en condiciones de temperatura , de humedad, presión atmosférica a nivel del mar.

Aplicando definiciones obtenemos que:

[[5]](#footnote-4)

1. **Otros conceptos utilizados:**

* ***“switch...case”* : *“****Al igual que las instrucciones* ***if, switch ... case*** *controla el flujo del programa permitiendo que los programas especifiquen un código diferente que debe ser ejecutado en distintas condiciones. En particular, una sentencia switch compara el valor de una variable con los valores especificados en las instrucciones case. Cuando se encuentra una sentencia case cuyo valor coincide con el de la variable, el código de esa declaración case se ejecuta.”[[6]](#footnote-5)*
* ***“break”*** : *“La palabra reservada break interrumpe la sentencia switch, y se usa normalmente al final de cada declaración case. Sin una declaración de interrupción, la sentencia switch continuará la ejecución de las siguientes expresiones hasta que se alcanza una interrupción o hasta el final de la sentencia switch.”[[7]](#footnote-6)*
* ***“delayMicroseconds()”*** : hace una pausa en el programa del tiempo especificado, como un *delay*, pero en microsegundos.
* ***“pulseIn()”*** : lee un pulso *HIGH o LOW* en un pin (funciona en los pulsos de 10 a 3 microsegundos).

1. **Resultados**

Uno de los resultados a los que llegamos es el código a utilizar:

//Conexiones del motor a la izquierda (abajo)

**#define** enA 11 //enable pin para el motor A

**#define** in1 10 //control pin 1 para el motor A

**#define** in2 9 //control pin 2 para el motor A

//Conexiones del motor de derecha (arriba)

**#define** enB 6 //enable pin para el motor B

**#define** in3 7 //control pin 1 para el motor B

**#define** in4 8 //control pin 2 para el motor B

//Sensor a la derecha

**int** TriggerD = 3;

**int** EchoD = 2;

//sensor a la izquierda

**int** TriggerI = 4;

**int** EchoI = 5;

//velocidad y distancia máxima que se puede "sensar" que fijamos

**int** vel = 200;

**int** dismax = 330;

**int** DURACIONd;

**int** DISTANCIAd;

**int** DURACIONi;

**int** DISTANCIAi;

**void** setup() {

Serial.begin(9600);

pinMode (enA, OUTPUT);

pinMode (enB, OUTPUT);

pinMode (in1, OUTPUT);

pinMode (in2, OUTPUT);

pinMode (in3, OUTPUT);

pinMode (in4, OUTPUT);

pinMode(TriggerD, OUTPUT);

pinMode(TriggerI, OUTPUT);

pinMode(EchoD, INPUT);

pinMode(EchoI, INPUT);

}

**void** lecturaSensoreSDER(){

//distancia de la derecha, o sea el sensor de la derecha

digitalWrite(TriggerD, HIGH);

delayMicroseconds(10);

digitalWrite(TriggerD, LOW);

DURACIONd = pulseIn(EchoD, HIGH);

DISTANCIAd = DURACIONd/ 58.27;

Serial.print("Distancia derecha: ");

Serial.println(DISTANCIAd);

delay(100);

}

**void** lecturaSensoreSIZQ(){

//distancia de la izquierda, o sea el sensor de la izquierda

digitalWrite(TriggerI, HIGH);

delayMicroseconds(10);

digitalWrite(TriggerI, LOW);

DURACIONi = pulseIn(EchoI, HIGH);

DISTANCIAi = DURACIONi/ 58.27;

Serial.print("Distancia izquierda: ");

Serial.println(DISTANCIAi);

delay(100);

}

**void** Adelante ()

{

//Dirección del motor de la izquierda

digitalWrite (in1, HIGH);

digitalWrite (in2, LOW);

analogWrite (enA, vel); //se le adjudica la velocidad establecida

//Dirección del motor de la derecha

digitalWrite(in3,HIGH);

digitalWrite(in4,LOW);

analogWrite (enB,vel); //se le adjudica la velocidad establecida

Serial.println("Adelante"); //Me escribe si va para adelante

}

**void** Atras ()

{

//Dirección del motor de la izquierda

digitalWrite (in1, LOW);

digitalWrite (in2, HIGH);

analogWrite (enA, vel); //se le adjudica la velocidad establecida

//Dirección del motor de la derecha

digitalWrite(in3,LOW);

digitalWrite(in4,HIGH);

analogWrite (enB,vel); //se le adjudica la velocidad establecida

Serial.println("Atras");//Me escribe si va para atrás

}

**void** Izquierda ()

{

//Dirección del motor de la izquierda

digitalWrite (in1, HIGH);

digitalWrite (in2, LOW);

analogWrite (enA, vel); //se le adjudica la velocidad establecida

//Dirección del motor de la derecha

digitalWrite(in3,LOW);

digitalWrite(in4,HIGH);

analogWrite (enB,vel); //se le adjudica la velocidad establecida

Serial.println("Izquierda");//Me escribe si va para la izquierda

}

**void** Derecha ()

{

//Dirección del motor de la izquierda

digitalWrite (in1, LOW);

digitalWrite (in2, HIGH);

analogWrite (enA, vel); //se le adjudica la velocidad establecida

//Dirección del motor de la derecha

digitalWrite(in3,HIGH);

digitalWrite(in4,LOW);

analogWrite (enB,vel); //se le adjudica la velocidad establecida

Serial.println("Derecha");//Me escribe si va para la derecha

}

**void** Detener ()

{

//Dirección del motor de la izquierda

digitalWrite (in1, LOW);

digitalWrite (in2, LOW);

analogWrite (enA, vel); //se le adjudica la velocidad establecida

//Dirección del motor de la derecha

digitalWrite(in3,LOW);

digitalWrite(in4,LOW);

analogWrite (enB,vel); //se le adjudica la velocidad establecida

Serial.println("Detener");//Me escribe si se detiene

}

**void** loop() {

**if** **(Serial.available())** //en caso de querer controlarlo por pequeños "impulsos" para ir en diferentes direcciónes

**switch** **(Serial.read())**//no sabemos si esta parte va para la entrega pero la dejamos por las dudas

{

case 'a':

Adelante();

break;

case 'b':

Atras();

break;

case 'c':

Izquierda();

break;

case 'd':

Derecha();

break;

case 's':

Detener();

break;

}

lecturaSensoreSDER(); //llmamos a las funciones de lectura de sensores

lecturaSensoreSIZQ();

**if**(DISTANCIAd < DISTANCIAi){

Derecha(); //el robotiño gira a la derecha si la distancia del sensor derecho es menor a la del izquierdo

}

**if**((DISTANCIAi>=dismax) && (DISTANCIAd<dismax)){

Derecha(); //el robotiño gira a la derecha si el sensor izquierdo no encuentra nada

}

**if**(DISTANCIAd > DISTANCIAi){

Izquierda();//el robotiño gira a la izquierda si la distancia del sensor izquierdo es menor a la del derecho

}

**if**((DISTANCIAd >= dismax)&& (DISTANCIAi<dismax)){

Izquierda();//el robotiño gira a la izquierda si el sensor derecho no encuentra nada

}

**if**(DISTANCIAd == DISTANCIAi){

Adelante();//si estan a la misma distanca van pa delante papu

}

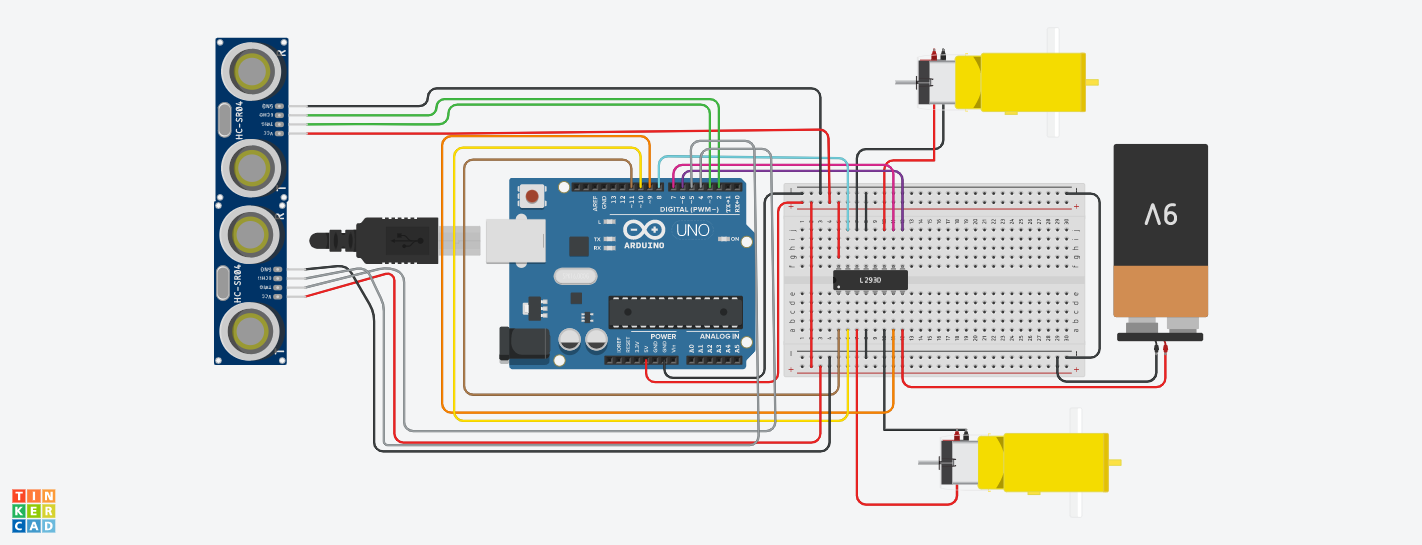
**if** ((DISTANCIAd >= dismax)&&(DISTANCIAi >= dismax)){

Detener(); //si no encuentran nada en el "radar" no se mueve, onda se detiene

}

}

Otro de los resultados es el circuito:[[8]](#footnote-7)



1. **Análisis de los resultados**
2. **Breve explicación del código:**

* En la primera parte de nuestro código definimos las constantes, en este caso serán los pines a los cuales están conectados los motores.
* Luego definimos los pines de los sensores, la velocidad, la distancia máxima que puede medir un sensor, la duración del pulso de cada sensor y las distancias a calcular que mide cada sensor, como variables *int* (que se usa para números enteros).
* Pasamos después a establecer el *setup()* del código, en este las únicas entradas *(INPUT)* serán los pines conectados al *Echo* de los sensores ya que el resto del código dependerá de este valor. En el *setup()* también comenzaremos la comunicación serial.
* A continuación definimos dos funciones diferentes (una para el sensor de la derecha y otra para el de la izquierda) para transformar el tiempo captado por los sensores en distancia. Lo primero que se hace es mandar un pulso de 10 microsegundos al *Trigger* del sensor, luego el pulso de respuesta por el *Echo* y medimos el pulso (la duración) con la función *pulseIn()*. Luego establecemos la relación entre el valor de la duración del pulso y la distancia para calcular a la misma. Lo que queda de la función es que imprima el valor de la distancia a través del uso del *Serial.println()*. Un dato importante a remarcar es que en esta función es de tipo *void()* porque en realidad no nos devuelve ningún valor, lo que hace es imprimirlo, en el caso de querer recibir un valor deberíamos utilizar un *return* junto con otro tipo de función.
* Luego establecemos como funciones individuales el movimiento de los motores (adelante, atrás, derecha, izquierda y detenido) de tipo *void()* (porque no regresan valores) en las cuales dependiendo del estado de los pines *(HIGH o LOW)* será el sentido de giro de los motores y consecuentemente la dirección a seguir por los mismos. Remarcamos que en los dos motores establecimos una velocidad que puede variar, optamos por ese valor (200) porque se encontraba entre el máximo y el mínimo que puede tomar un motor (0 - 255).
* Al final llegamos al *loop()* donde establecemos una relación entre un char y una función, o sea si mandamos un *“a”* por el monitor serial la función que “actuará” será la de *Adelante()* y así con el resto de las funciones de dirección. Esta parte la pusimos debido a que no sabíamos si se debía incluir.
* Después pasamos a la parte “esencial” del ejercicio, donde hacemos depender de la posición de un objeto a la dirección que seguirá el robot. Lo que hicimos fue (luego de “invocar” las funciones de lectura de sensores en el *loop()*) establecer los diferentes tipos de situaciones con los que podamos llegar a encontrarnos a la hora de tener un objeto en frente, estas son:

1. Caso en el cual tenga dos valores (las distancias tanto de la derecha como de la izquierda pueden ser captadas).
2. Si la distancia de la derecha es menor que la de la izquierda, se encontrará más cerca del objeto por su parte derecha, por lo cual deberá avanzar hacia la derecha.
3. Si la distancia de la izquierda es menor a la de la derecha, se encontrara más cerca del objeto por su parte izquierda, por lo cual avanzará hacia la izquierda.
4. Caso que una de las distancias no pueda ser captada (si pasan el valor máximo ni se pueden leer).
5. Si la distancia de la izquierda es mayor o igual al valor máximo, pero la distancia de la derecha se puede captar (es menor al valor máximo), el robot se moverá hacia la derecha.
6. Si la distancia de la derecha es mayor o igual al valor máximo, pero la distancia de la izquierda se puede captar (es menor al valor máximo), el robot se moverá hacia la izquierda.
7. En el caso que se encuentre a la misma distancia en ambos sensores se moverá hacia adelante.
8. En el caso de que los sensores no puedan captar nada el robot no se moverá ( a menos que se le mande una orden).
9. **Problemas encontrados:**

Al aplicar el código al circuito realizado nos encontramos con ciertas limitaciones físicas que predisponen el resultado, haciéndolo variar de lo teóricamente esperado. Algunas de estas diferencias son:

1. **Las distancias obtenidas:** al variar la distancia del objeto sensado en Tinkercad, podemos darnos cuenta de una pequeña diferencia en relación a la distancia obtenida por el código y el valor de distancia que muestra el sensor (si lo clickeamos).
2. La primera conjetura que se maneja es que Tinkercad marca la distancia al objeto incluyendo la zona muerta de los sensores. Una zona muerta se refiere al área directamente delante de la cara del transductor, donde el sensor no puede realizar mediciones confiables
3. La segunda es que debido a la utilización de datos como la velocidad del sonido, la conversión de esta a la medidas específicas, hace que las medidas finales pierdan exactitud.
4. **Superposición de los campos de actuación:** una de las limitantes encontradas, es que para que un objeto sea sensado con dos “coordenadas” (o sea tenga un valor en relación al sensor derecho e izquierdo) deba encontrarse en el área superpuesta de los campos de actuación de cada uno.
5. **Distancia máxima sensada:** uno de los valores que utilizamos para poder determinar el accionar del robot es la máxima distancia que un sensor puede medir. En este caso encontramos diferentes valores para esta distancia en internet (4m o 5m) pero estos no funcionaban como queríamos en Tinkercad, por lo cual a modo de acierto y error determinamos la distancia máxima moviendo un objeto por el campo de actuación hasta llegar al límite.
6. **Conclusiones**

De esta experiencia recalcamos la importancia de la comunicación serial para mandar ciertas “órdenes” tanto como para mantenerse informado del funcionamiento de los circuitos, en otras palabras controlar el mismo a través de la llegada de valores esperados, en el caso de presentar una irregularidad o un error estos probablemente se visualizarán en el monitor serial.

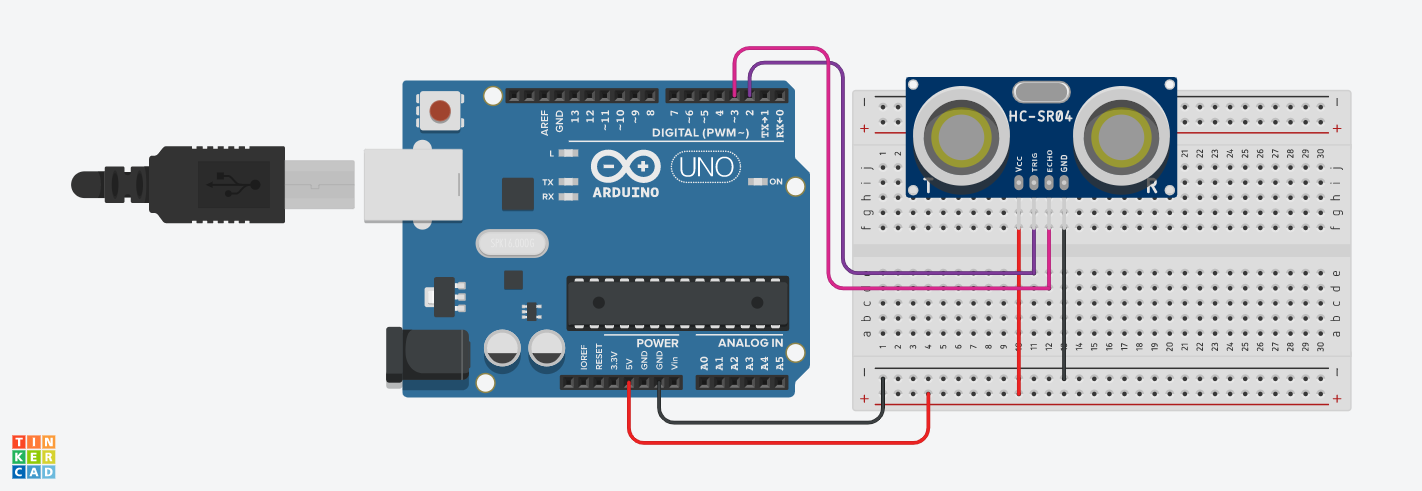
En relación a los sensores ultrasónicos, a pesar de las restricciones que presentan estos son bastante accesibles a la hora de programar.

1. **Referencias**

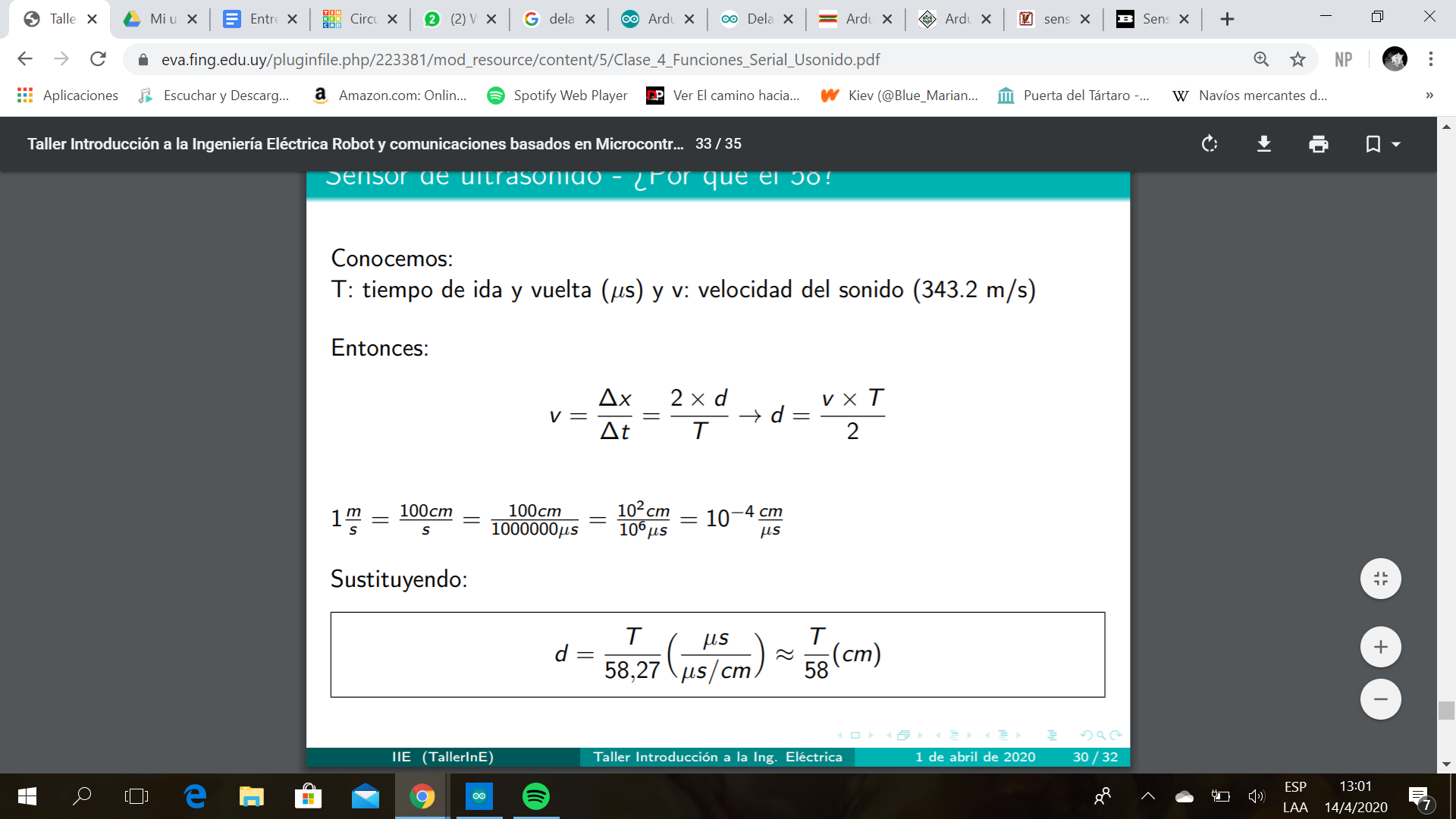
* http://manueldelgadocrespo.blogspot.com/p/serialavailable.html
* http://manueldelgadocrespo.blogspot.com/p/instruccion-swi.html
* https://descubrearduino.com/
* <https://proyectoarduino.com/sensor-de-ultrasonidos-medir-distancia-con-arduino/>
* <https://descubrearduino.com/dc-motor/>
* https://www.prometec.net/programa-varias-funciones/
* http://cursoarduino.proserquisa.com/2016/10/05/tutorial-6-que-es-una-funcion-y-como-utilizarla/

1. **Anexo**

**Imagen I**

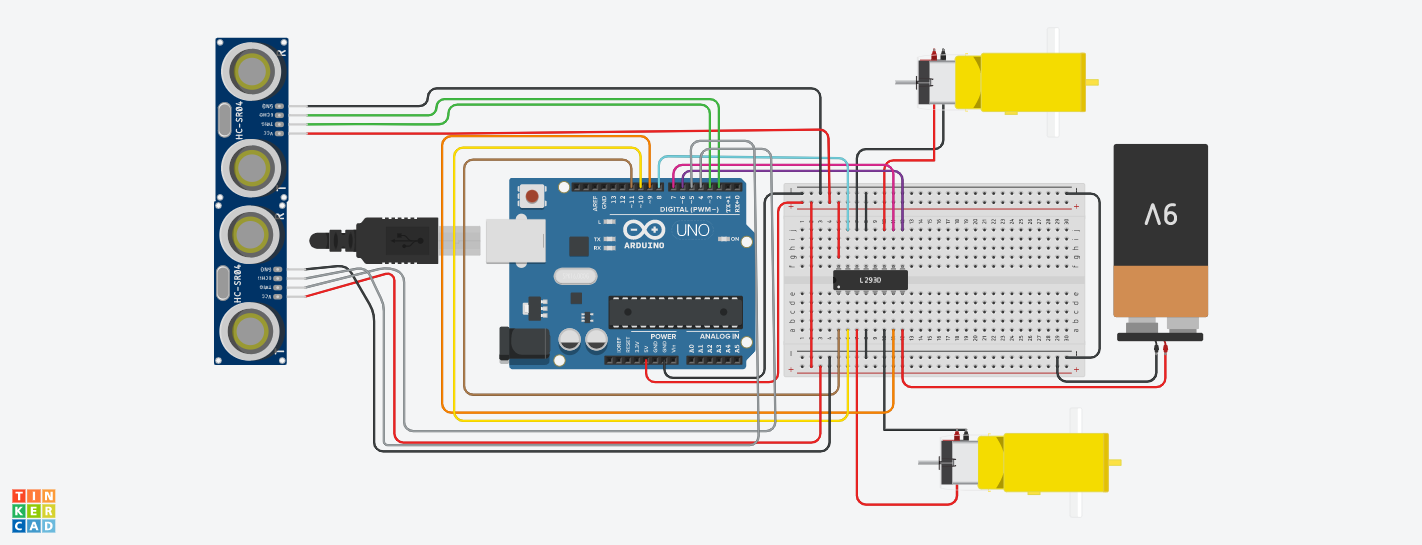
****

**Imagen II**



*(Extraída de la presentación de la clase 5 TallerInE)*

**Imagen III**

****

1. Motor de corriente continua [↑](#footnote-ref-0)
2. Un puente en H es un circuito electrónico que puede accionar el motor en ambas direcciones. [↑](#footnote-ref-1)
3. El buffer tiene una capacidad de almacenamiento de 128 bytes. [↑](#footnote-ref-2)
4. Ver imágen **I** en el anexo del circuito de un sensor ultrasónico conectado a una placa Arduino. [↑](#footnote-ref-3)
5. Por más información de la transformación ver imágen **II** en el anexo. [↑](#footnote-ref-4)
6. *(13/4/20). switch...case. Lugar de publicación: Arduino Español. http://manueldelgadocrespo.blogspot.com/p/instruccion-swi.html* [↑](#footnote-ref-5)
7. *Ídem.* [↑](#footnote-ref-6)
8. Imagen **III** en el anexo (imagen ampliada del circuito) [↑](#footnote-ref-7)