Proyecto mini radar

Alumno/a: Dayanna Vila

Profesor: Gonzalo Consorti.

Integrantes: Dayanna Vila, Thiago Arellano, Alejandro Alarcon y Felipe Luchetta.

Curso: 4°1 TT

Ciclo Lectivo: 2024

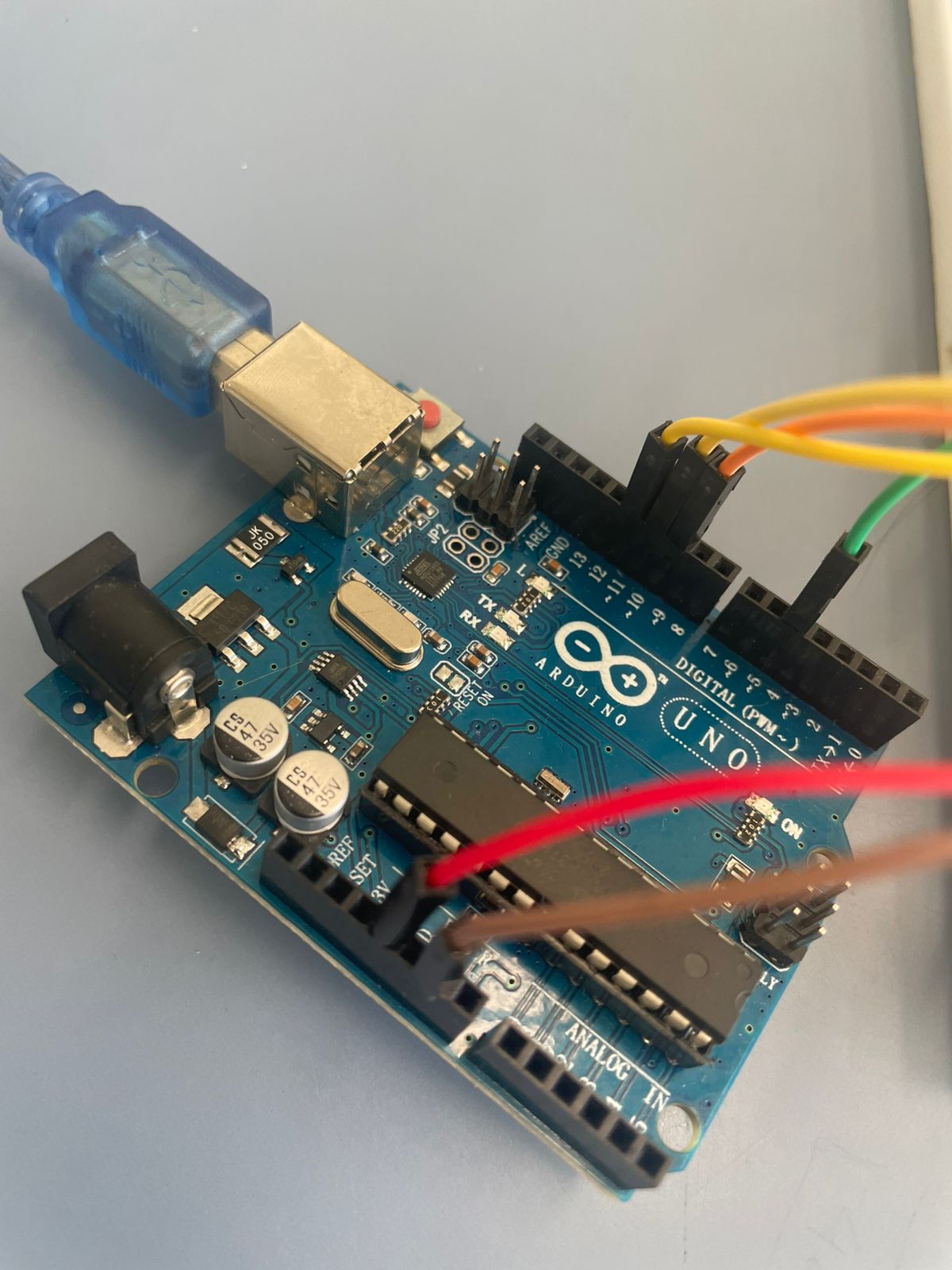


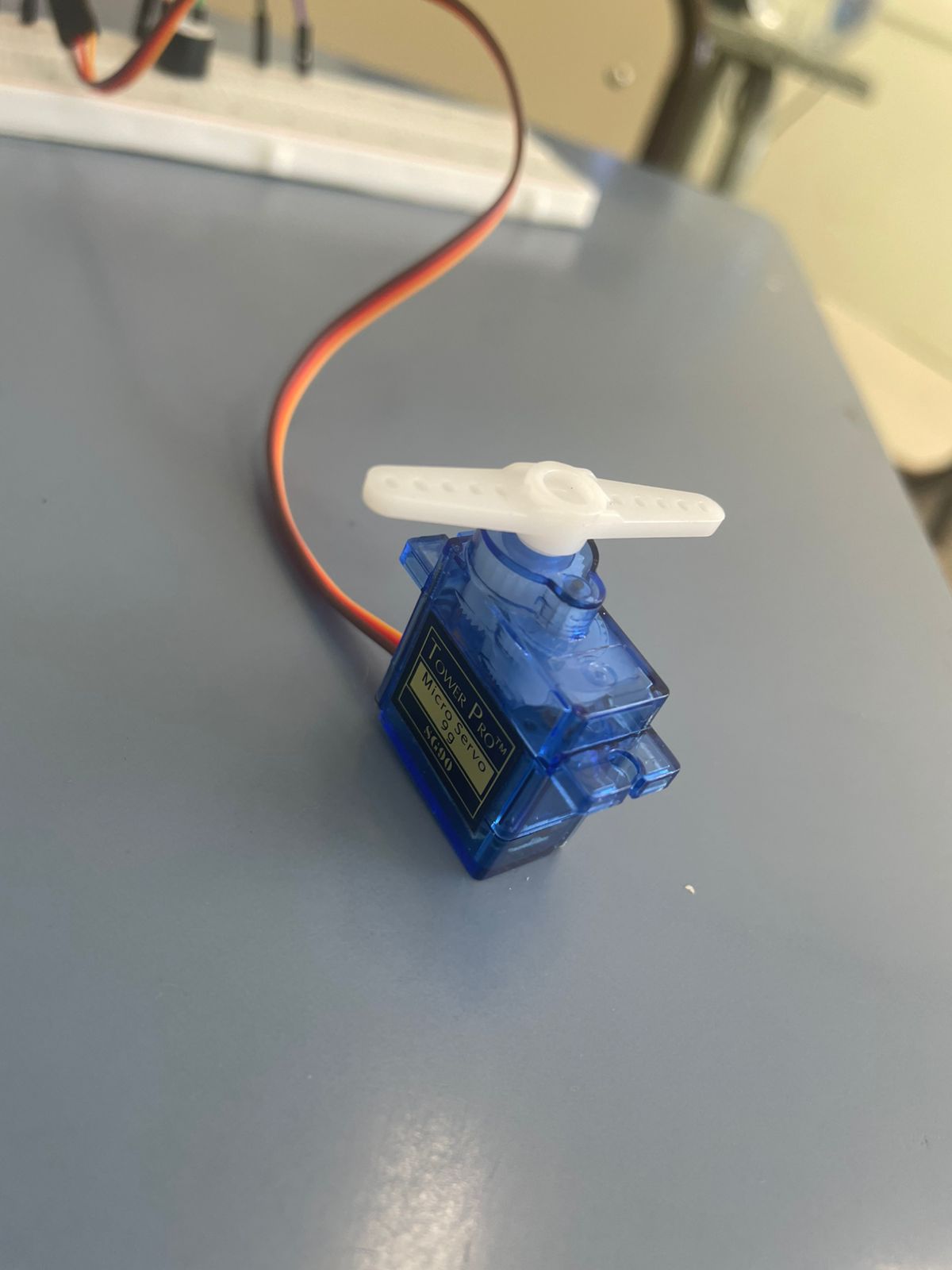
**Introducción**

Con nuestro grupo elegimos hacer el mini radar porque era el menos “complicado” de los que quedaban. Nos dividimos el trabajo, ya que aparte de trabajar con Arduino también tenemos que trabajar con Arduino processing. Hay que lograr hacer un mini radar que pueda captar movimientos o alguna presencia de un objeto y lo muestre en pantalla. Para esto hay que buscar información.

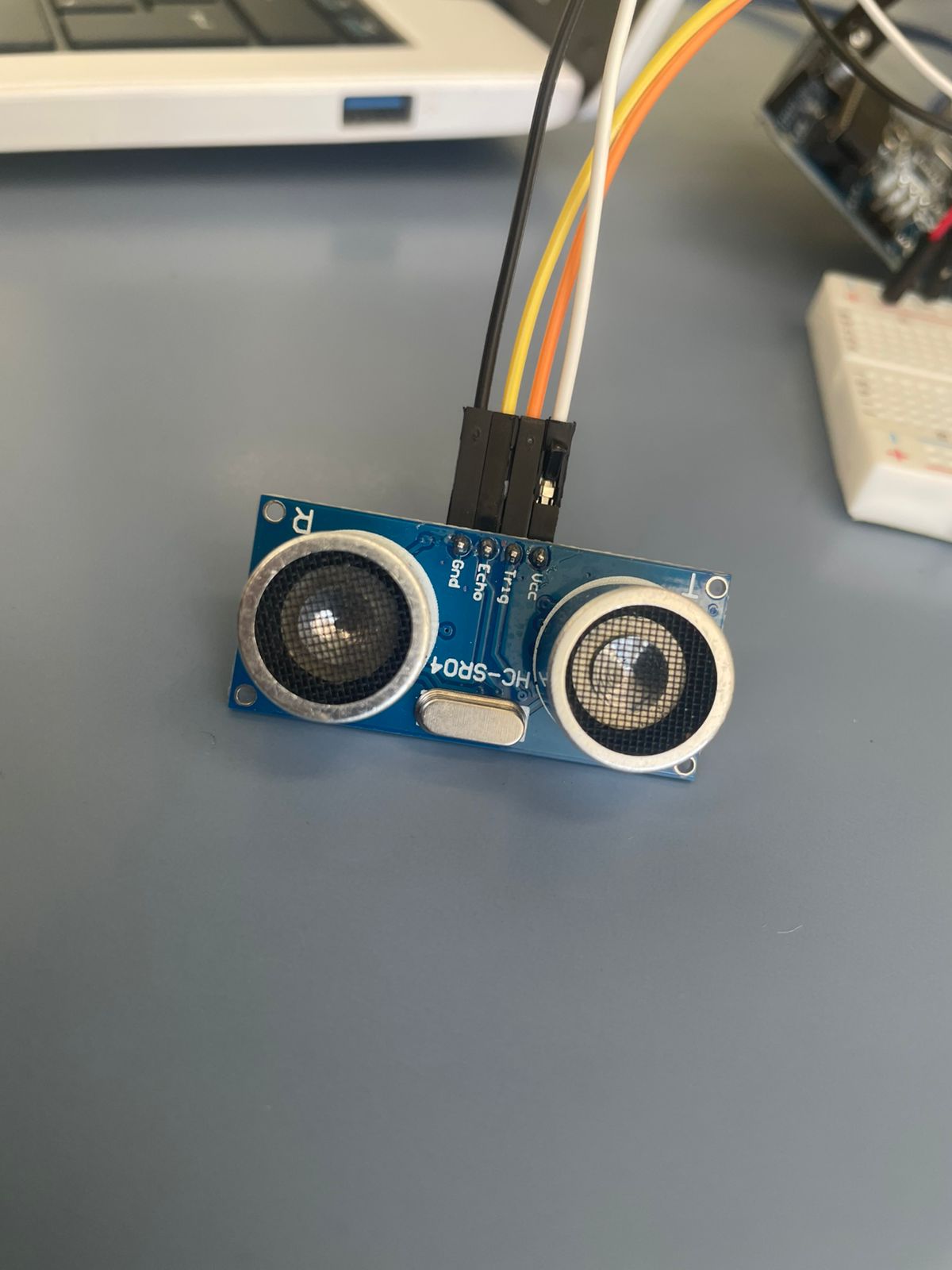
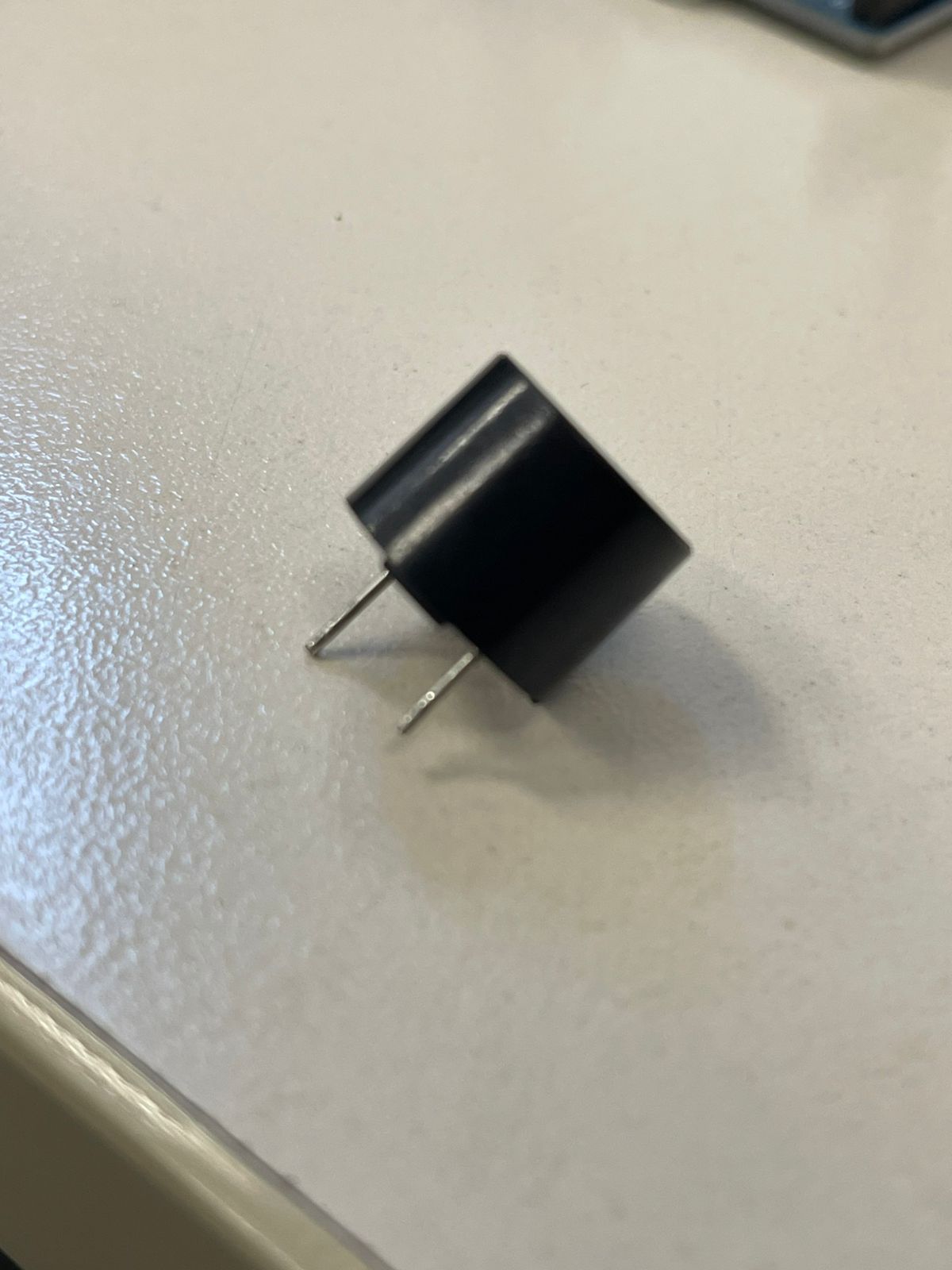
**Componentes**

Para el proyecto necesitamos los siguientes componentes:

Placa de Arduino UNO: Es el cerebro, cabeza,corazón de nuestro proyecto, va a ser el responsable de realizar todos los procesos a través del código. 

Servo motor: Lo necesitaremos para mover el sensor para que pueda escanear el área en diferentes ángulos haciendo rotar el sensor del radar.

Sensor ultrasónico: Va a ser el encargado de detectar la distancia o los objetos cercanos dentro de una distancia determinada, esto hará que si detecta una distancia por ejemplo a 20cm el buzzer pueda sonar.

Buzzer: Vamos a utilizar el buzzer como alarma. Cuando el sensor detecte algún movimiento o presencia, el buzzer va a ser el encargado de avisarnos.

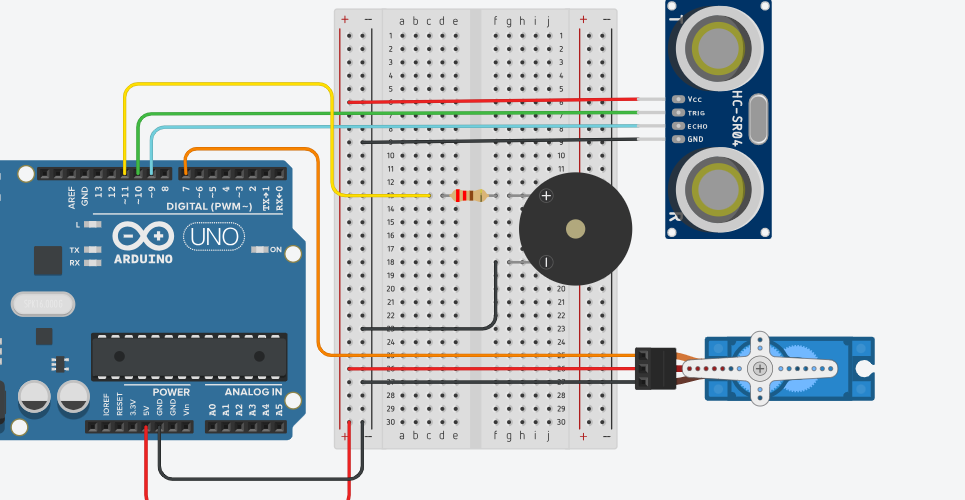
También necesitaremos una protoboard para no hacernos tanto los cables(alerta spoiler hubo lío igual).

Para hacer el proyecto vamos a necesitar un [soporte](https://cults3d.com/es/modelo-3d/artilugios/sr04-servo-mount-bugz). Estaba pensado lo de poner un soporte así el sensor da vuelta junto con el radar, pero gracias a que alguien mandó tarde a imprimir no creo que lleguemos con el soporte ya que lo teníamos que mandar a imprimir.

El radar es un dispositivo que utiliza ondas de radio para medir distintas características de los objetos, como la velocidad, la distancia, cualquier tipo de movimiento. Su función es captar movimientos dentro de su lectura de campo. Existen distintos tipos de radares, en nuestro caso nosotros vamos a hacer un radar fijo que detecte movimiento y lo muestre en la pantalla.

**Esquema de Conexiones**

Para no quemar los componentes y que Consorti no nos mate, vamos a practicar con tinkercad las conexiones.Esta fue nuestra base:



[Nos basamos en imagenes](https://images.app.goo.gl/CcZQ6P9VjoScesYS9) para poder hacer mejor las conexiones.

El resultado final fue el siguiente:

A nuestro parecer la conexión de cables es mucho mejor que las imagenes, porque se ve un orden.

Vamos a utilizar el pin 10 que se va a encargar de mandar un pulso de ultrasonidos, y el pin 9 se va a encargar de recibirlo. Para la conexión al servomotor vamos a utilizar el pin 7.

Igual esto es un ejemplo o guía para lo que vamos a hacer, porque el sensor tendría que ir pegado al servomotor junto con el soporte que hay que imprimir en 3D. También necesitamos poner todo en una base de cartón y madera para que el proyecto quede inmovil.

**Servomotor**

Para usar el [servomotor](https://youtu.be/MaBl8ibDvZ8?si=li3lxVNaxMpu7Xs8)  hay que tener en cuenta que tiene 3 pines digitales, uno que va conectado a tierra(cable negro), otro a la potencia(cable rojo) y el último que es el que va a recibir la señal del arduino(cable naranja). Para el código tenemos que incluir la [biblioteca servo](https://arduinogetstarted.com/es/reference/library/arduino-servo-library) para controlar el servomotor, para ello utilizaremos el siguiente código: #include <Servo.h> (estamos incluyendo la biblioteca servo). El servo contiene distintas funciones:

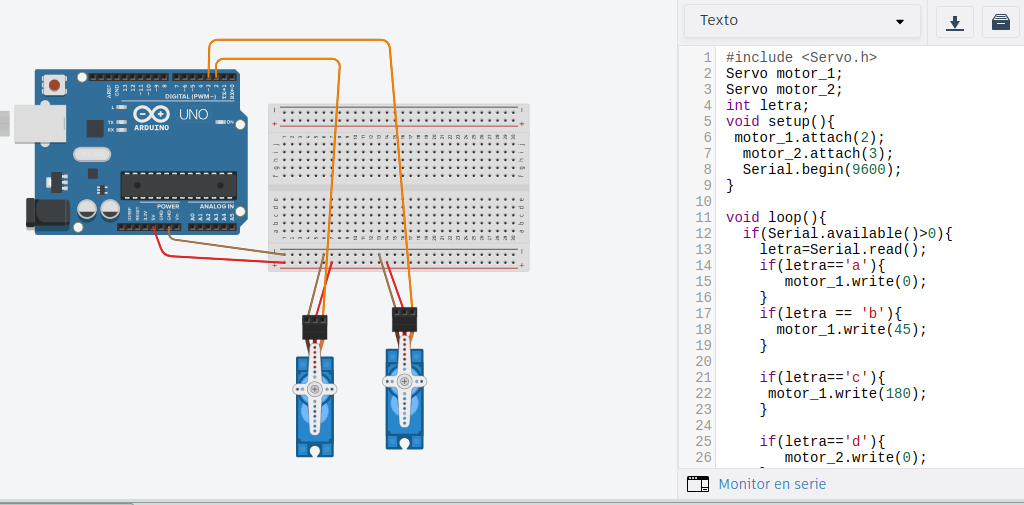
Servo.attach(): Conecta el pin que utilicemos a nuestro servo.

Servo.write():Indica el ángulo al que posicionamos el servomotor.

Servo.writeMicroseconds(): Indica la anchura de pulso (en microsegundos) al servo.

Servo.read(): Devuelve el ángulo en donde se encuentra el servo.

Servo.dettach(): Indica al código que libere el servo y deje de comunicarse con él.

Como ejemplo para usar el servomotor usamos un trabajo práctico que tuvimos junto con mi compañero(Alejandro Alarcon). Para este TP teníamos que utilizar dos servomotores y hacer que cuando se pone una letra en el monitor en serie el servo cambie de ángulo(dependiendo la letra).

**Fuentes**

[POYECTO RADAR DE ROBOT UNO](https://youtu.be/tDJb5aCa9PQ?si=4hxg2Jd497iVjP1e)

Desde el minuto 1:0 hasta 4:32 nos explica el montaje del proyecto, aunque estén todos los cables desordenados lo de poner una tabla de madera para el proyecto es una buena idea(de cartón también podría ser una opción) .En el minuto 4:34 nos enseña la programación en Arduino IDE hasta el minuto 7:12 que se enseña como funciona el proyecto. En mi opinión el video es bueno en montaje y explicando el código del arduino pero faltaría explicar el código de processing.

[PROYECTO RADAR DE CARLOS SUAREZ](https://youtu.be/RaVIzulODQ0?si=HCus1rzcQ9pz29gt)

El video te explica el código(no tanto de processing) para usarlo de base es bueno. También utilizando una librería del módulo de ultrasonido. Al igual que el otro video nos deja ya hecho el código de processing.

<https://codemain.pro/radar-con-arduino-y-processing/>

<https://eloctavobit.com/proyectos-tutoriales-arduino/realizar-un-radar-con-arduino-y-processing> Ambos enlaces nos dan códigos de processing, el segundo enlace nos explica ambos códigos paso a paso. Para nosotros es importante aprender a utilizar el de processing ya que tenemos que empezar de cero.

Como en la mayoría de fuentes nos dan directamente los códigos y algunos tienen los mismos códigos tendremos que investigar para qué es cada código y basado en todo eso haremos nuestro propio código de processing.

Con todo lo aprendido en clase más los videos ya teníamos una idea de como hacer el código. Para no complicarnos la vida vamos a sacar el tiempo como nosotros sabemos, NO vamos a utilizar la biblioteca HC-SR04.

Para empezar con el código es necesario incluir la biblioteca servo, así le indicamos que vamos a usar el servomotor. Luego lo declaramos con Servo myservo;

Después vamos a definir los nombres de los pines ; 10 que es el que está conectado al desencadenador,lo vamos a definir como trig. Y el 9 que está conectado al Eco como echo

Dentro del void setup va a ir el radar.attach(7);

que como se mencionó anteriormente va conectar el pin al servo.

Para el buzzer que es el que nos va a avisar si detecta algún objeto, lo vamos a conectar al pin 7.

**Código de arduino**

Dentro del void loop() usamos el bucle for para que el servomotor se mueva a 180°. Y el radar.write() para indicar a qué ángulo se tiene que mover el servo. Así quedaría el código

| for (int i = 180; i > 15; i--) {  myServo.write(i);   delay(30);   distancia = calcularDistancia(); |
| --- |

También dentro del for vamos a enviar señal del disparo al sensor. A continuación vamos a crear dos variables:

Duración: que va almacenar el tiempo en el que el echo tarda en regresar al objeto.El pulseIn lo utilizamos para medir el tiempo en el que el pin echo este en estado HIGH.

Distancia: que va a ser la duración\*10/292/2 para almacenar la distancia medida al objeto en centímetros.

Ya que tenemos la distancia vamos a instanciar con un if que si el sensor detecta movimiento menor a 50cm se va activar el buzzer y un mensaje en el monitor. En caso de no detectar nada utilizaremos el else para que el buzzer esté apagado.

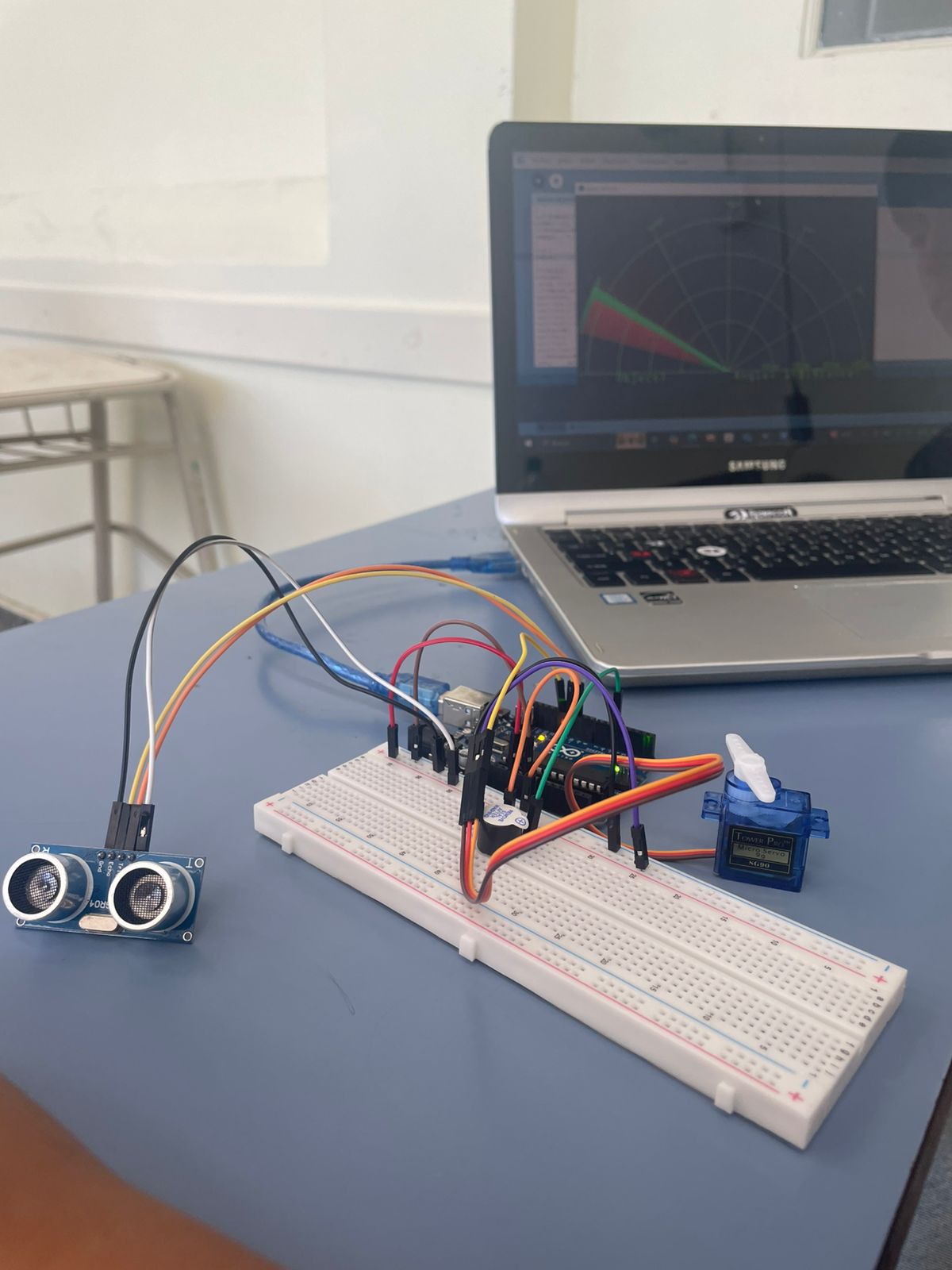
| if (distancia < 10) {  tone(5, 500, 100);   } else if (distance > 20 && distancia < 40) {  tone(5, 200, 100);  } |
| --- |

Fue complicado hacer el código fueron varios los errores por ejemplo: Quise instanciar el if y el else dentro del for, pero no declare la distancia dentro de este lo cual me mostraba que la distancia no estaba definida, faltaba enviar la señal al desencadenador y también faltaba ordenar los corchetes para saber dónde termina cada código. Fueron horas estresantes para entender el error porque estaba mal ordenado.

Ahora logramos terminar el código en tinkercad, tendríamos que probarlo en arduino con los componentes.

Quedó de la siguiente manera

Tuvimos algunos problemas para empezar a hacer el proyecto y es que a medida que queríamos avanzar siempre faltaba un componente o destruiamos el proyecto de alguien más ya que entre 3 cursos utilizabamos los mismos componentes, fue un bajón porque si no sacabamos foto era empezar con el montaje de 0. Pero gracias a que mi compañero Alejando Alarcon que compró los componentes avanzamos una banda, no es por nada pero mínimo que eso le suba un punto.

Aca esta el proyecto armado(ignoremos el desorden de cables) lo probamos con el código y funcionaba correctamente bueno uno que otro error pero de pines ya que yo el código lo hice en tinkercad y al armarlo lo cambiamos. El hacer el código fue fácil ya que me base en trabajos prácticos que ya habíamos hecho por ejemplo el tp5(no en todo solo en la parte del sensor) y la clase 6 de vectores para usar el for junto con el servomotor, y el servo ya sabia usarlo por trabajos que tenía de segundo año con el profesor Nahuel Acosta.

Si pensé que ya habíamos terminado fue todo lo contrario, recién empezamos con lo complicado de entender processing .

**Processing**

Arduino processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en JAVA.

Este programa está diseñado para diseñadores audiovisuales quienes crean proyectos audiovisuales o multimedia, ya que se pueden realizar gráficos 2D, 3D, texturas,etc.

Los códigos son similares a los que trabajamos con Arduino, la estructura del sketch es muy similar. Tenemos una función setup() donde iniciaremos las variables de nuestro proyecto y tendremos una función similar al loop (que es el que utilizamos en Arduino) pero en processing se llama draw siendo este un bucle infinitivo.

En la primera línea de código comenzamos importando la librería que necesitamos para la comunicación serial. Se importa de la siguiente manera ; import processing.serial\*; Esta librería es fundamental ya que es la que va a interactuar con el puerto en serie y va a ser el encargado de recibir datos enviados desde el arduino. Declararemos ‘puerto’ como objeto para manejar la conexión en serie.

Se tienen que crear variables para mantener los valores del ángulo y de la distancia:

String SerialAngulo : Almacenaremos el ángulo recibido.

String SerialDistancia : Almacenará la distancia recibida.

String SerialDatos:Guardará los datos recibidos en una sola cadena.

float ObjetoDistancia: Almacena la distancia del objeto con números decimales.

int radarAngulo, radarDistancia: Variable que nos servirá para almacenar los valores de ángulo y distancia en enteros.

int index= 0: Variable para almacenar el índice de separación entre el ángulo y la distancia en los datos recibidos.

En la función setup() vamos a indicar el tamaño de la ventana gráfica utilizando size(), dentro vamos a determinar del tamaño que va a ser la pantalla del radar. Comenzaremos la comunicación indicando el puerto que estamos utilizando. Y si utiliza [smooth()](https://static.uvq.edu.ar/mdm/paam/unidad-02-06.html#:~:text=Las%20funciones%20smooth()%20y,cada%20uno%20de%20los%20objetos.) y noSmooth() activa o desactiva las opciones de suavizado de la representación en cada objeto.

La funcion [buffer](https://www.luisllamas.es/buffer-circular-arduino/) : nos permite almacenar los últimos N elementos en una array de longitud N almacenaremos los datos recibidos desde el Arduino hasta que se encuentre un símbolo específico, nosotros utilizaremos ‘#’. Este símbolo va a tener la función de indicar el final del conjunto de datos. Al enviar datos desde el Arduino se envía el ángulo del radar seguido por un \* , luego la distancia medida y finalmente se termina la transmisión con el símbolo. Esto hace que processing identifique cuando empieza y termina cada mensaje.

La función [serial.Event()](https://www.arduino.cc/reference/es/language/functions/communication/serial/serialevent/)es la encargada de manejar la llegada de nuevos datos. Esta función se activa automáticamente cuando hay datos disponibles en el puerto serial. puerto.[readString('#')](https://search.arduino.cc/search?tab=reference&q=readstring) : sirve para leer todos los datos hasta que se encuentre el símbolo. Esto será almacenado en una variable llamada SerialDatos. Después de haber leído los datos.

Usaremos [substring()](https://developer.mozilla.org/es/docs/Web/XPath/Functions/substring) para eliminar el último carácter que sería ‘#’ ya que no queremos incluirlo en los datos procesados. Como los datos contienen el ángulo y la distancia se busca la posición del asterisco para separar ambos elementos.

La función [indexOf(‘\*’)](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf) devuelve la posición del asterisco dentro de la cadena.

Utilizando substring() extraemos los valores correspondientes al ángulo y distancia. Finalmente ambos valores son convertidos a enteros y se almacenan en las variables radarAngulo y radarDistancia, que seran utilizados posteriormente para dibujar las líneas en pantalla.

La función [draw()](https://p5js.org/es/reference/p5/draw/) es donde ocurre el proceso visual del radar, es el responsable de actualizar lo que se nos muestra en pantalla. Utilizando los valores almacenados en radarAngulo processing dibujara una línea que representa la dirección del pulso emitido por el sensor ultrasónico. A medida que el servomotor gira y el sensor lea su campo las líneas son dibujadas en tiempo real por pulso. Esto hace que se pueda visualizar cómo cambia según la detección de movimientos. A medida que el radar vaya girando se van a dibujar las líneas dependiendo el pulso en tiempo real.

Existen dos funciones para definir el grosor del borde:

[strokeWeight()](https://processing-spain.blogspot.com/2015/09/312-definir-el-grosor-del-borde.html) es la que define las líneas/grosor de la forma que se van a ver en pantalla, [noStroke()](https://processing-spain.blogspot.com/2015/09/312-definir-el-grosor-del-borde.html) al contrario del otro este es para no mostrar las líneas para que se vea la forma sin bordes.

El comando stroke() nos permite poner el grosor del borde.

El [fill()](https://sites.google.com/site/proyectosbarron/simuladores/animaciones-con-processing/comandos-fill-y-stroke) nos permite poner colores , el [nofill()](http://jvaraujo.art/programacioncreativa/capitulo-1/11-juguemos/) elimina el relleno, el arc define un arco() en dirección a las agujas del reloj y el line(x1,y1,x2,y2) es un conjunto de funciones para dibujar formas. Para hacer la pantalla del radar rectangular se usa [rect()](https://processing-spain.blogspot.com/2015/09/36-dibujar-un-rectangulo-rectx-y-width.html). [Starline()](https://processing.org/examples/star)dibuja una amplia gama de diferentes formas. La función [Height](https://p5js.org/es/reference/p5/height/#:~:text=La%20variable%20height%20almacena%20la,valor%20de%20height%20como%2050.) almacena la altura de la ventana de visualización.

La función [pushmatrix()](https://processing.org/reference/pushMatrix_.html) guardará la transformación actual (osea la posición y rotación) en una pila y popmatrix() restaura las coordenadas.

Como necesitamos que el sistema mueva el origen del sistema de coordenadas al punto usaremos [traslate()](http://www.mywonderland.es/curso_js/processing/pro_trans.html#:~:text=Para%20rotar%20un%20objeto%20en,()%20seguido%20de%20rotate().).

Para explicar todo esto me base en una de las fuentes que nos mostraba un codigo entonces me puse a investigar la función de cada codigo.Con toda esta información nos ponemos manos a la obra para la instancia final del proyecto.

Al principio cometimos muchos errores para el código, probamos con distintas fuentes que nos pasaron su código , en uno el radar no corría directamente la barra se quedaba quieta ni se movía y en los demás no funcionaban. Bueno como no funcionan vamos a tomar inspiración en ellas, y ya que buscamos para que sirve cada función vamos a hacerlo poco a poco.

**Código Processing**

Dentro del void setup vamos a establecer el tamaño de la pantalla que va a ser de 1200 pixeles de ancho por 700 pixeles de alto, y vamos a mejorar los bordes usando smooth. Luego iniciaremos la comunicación en el puerto COM3, se almacenarán los datos hasta recibir el punto.

| void setup() {  size(1200, 700);  smooth();  miPuerto = new Serial(this, "COM3", 9600);  miPuerto.bufferUntil('.'); } |
| --- |

| void draw() {  fill(98, 245, 31);  noStroke();  fill(0, 4);  rect(0, 0, width, height - height \* 0.065);  fill(98, 245, 31);  dibujarRadar();  dibujarLinea();  dibujarObjeto();  dibujarTexto(); } |
| --- |

En el void draw utilizaremos fill() que nos servirá para hacer el rectángulo y fondo del radar. Luego llamaremos a varias funciones que dibujan diferentes elementos dentro de la pantalla para dibujar el gráfico, la línea que va a representar el ángulo actual, una línea que representa la distancia al objeto detectado y la última función va a ser para dibujar la información textual sobre el objeto detectado.

Utilizaremos un void serialEvent que se activará cuando reciba nuevos datos a través del puerto en serie, se leerán los datos hasta encontrar el (.) y se almacenarán en la variable datos. Luego haremos otra función que elimine el (.) y se busque la posición de la (,) para separar el ángulo de la distancia. Se extraerán las subcadenas correspondientes al ángulo y la distancia para convertirse en enteros.

| void serialEvent(Serial miPuerto) {  datos = miPuerto.readStringUntil('.');  datos = datos.substring(0, datos.length() - 1);  indice1 = datos.indexOf(",");  angulo = datos.substring(0, indice1);  distancia = datos.substring(indice1 + 1, datos.length());   iAngulo = int(angulo);  iDistancia = int(distancia); } |
| --- |

Tuvimos un error en esta funcion que fue la siguiente:

datos= datos.substring(0, datos.length() + 1());

Esto provoca un error ya que datos.length() +1()); supera el índice máximo de la cadena. La solución es:

datos= datos.substring(0, datos.length() - 1());

Aca evitamos ir fuera del índice máximo y eliminamos el último carácter.

| void dibujarRadar() {  pushMatrix();  translate(width / 2, height - height \* 0.074);  noFill();  strokeWeight(2);  stroke(98, 245, 31);  arc(0, 0, (width - width \* 0.0625), (width - width \* 0.0625), PI, TWO\_PI);  //más código…   line(-width / 2, 0, width / 2, 0);  line(0, 0, (-width / 2) \* cos(radians(30)), (-width / 2) \* sin(radians(30)));  //más código... } |
| --- |

En la función void dibujarRadar se va a dibujar el gráfico circular que representa el radar. Dentro de este utilizaremos pushMatrix y pop Matrix para guardar y restaurar las transformaciones gráficas y el traslate será el encargado de rotar el radar. Con el arc(), line() se dibujarán arcos que representan diferentes distancias y líneas que nos indican los ángulos relevantes, el width será para centrar los arcos y líneas del radar. Usaremos radians para calcular ángulos y posiciones en la visualización de datos.

Luego haremos una función para dibujar la línea roja que va a representar la posición del objeto detectado basándose en el ángulo y la distancia medidas, esto se activará si la distancia es menor a 40 cm.

| void dibujarObjeto() {  pushMatrix();  translate(width / 2, height - height \* 0.074);  strokeWeight(9);  stroke(255, 10, 10);  pixelesDistancia = iDistancia \* ((height - height \* 0.1666) \* 0.025);  if (iDistancia < 40) {  line(pixelesDistancia \* cos(radians(iAngulo)), -pixelesDistancia \* sin(radians(iAngulo)), (width - width \* 0.505) \* cos(radians(iAngulo)), -(width - width \* 0.505) \* sin(radians(iAngulo)));  }  popMatrix(); } |
| --- |

La función void dibujarLinea se encargará de dibujar una línea verde que indique el ángulo actual del sensor respecto al eje horizontal. Utilizaremos la funciones traslate, line, width y heigth para hacer la parte verde del radar.

| void dibujarLinea() {  pushMatrix();  strokeWeight(9);  stroke(30, 250, 60);  translate(width / 2, height - height \* 0.074);  line(0, 0, (height - height \* 0.12) \* cos(radians(iAngulo)), -(height - height \* 0.12) \* sin(radians(iAngulo)));  popMatrix(); } |
| --- |

Void dibujartexto que es la última función tiene como objetivo mostrar información textual en la pantalla. Esta información incluye la distancia medida por el sensor y una indicación de si se detectó algún objeto.

Ya tenemos el proyecto terminado a pesar de los días estresantes logramos hacer que nuestro mini radar pueda captar movimiento y lo muestre en pantalla. El proceso fue estresante y el resultado valió la pena.

Como todo grupo tuvimos problemas en cosas mínimas y demás… Pero fue lindo ver como el radar funcionaba y se mostraba todo en pantalla.