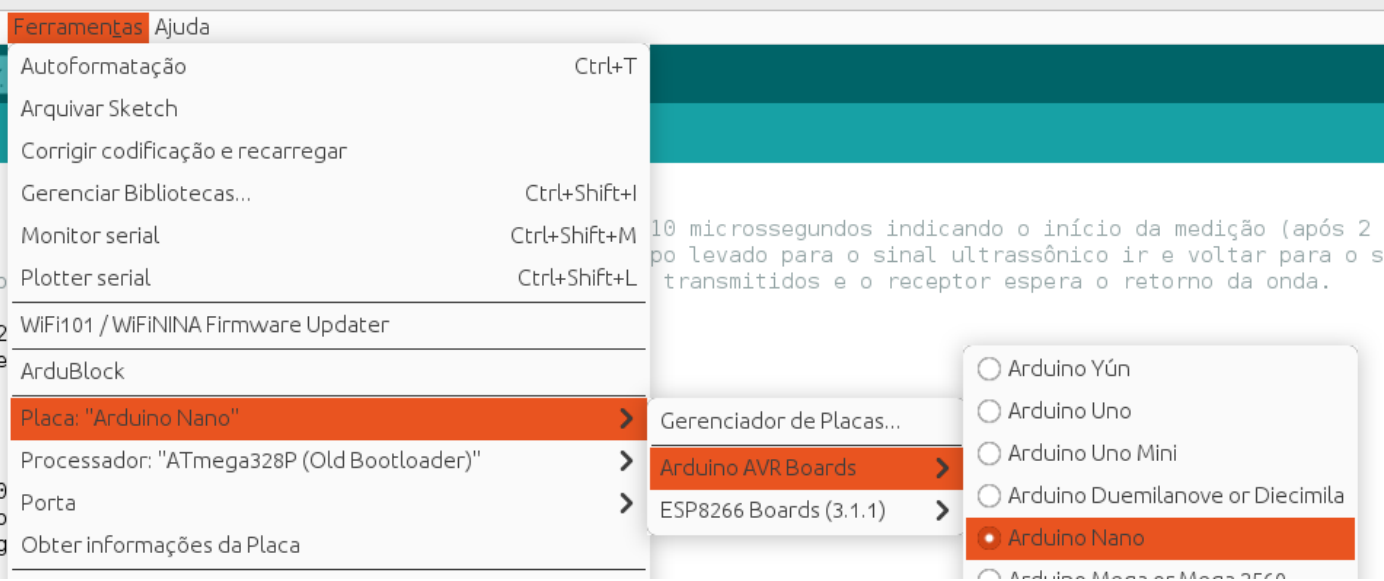
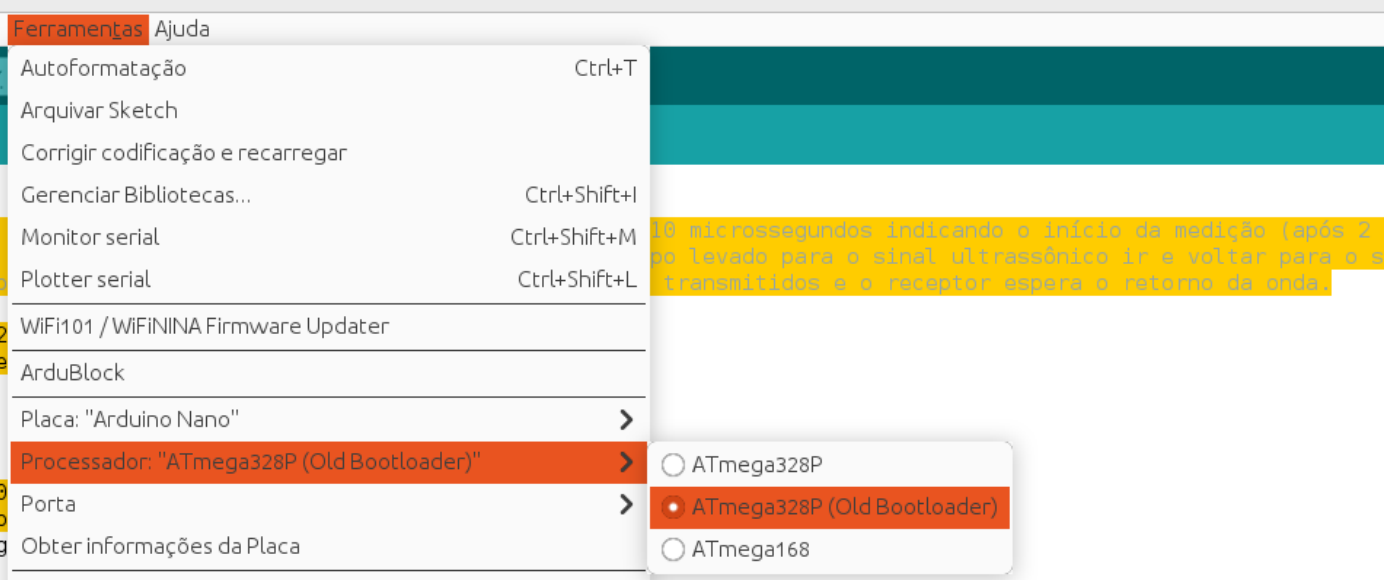


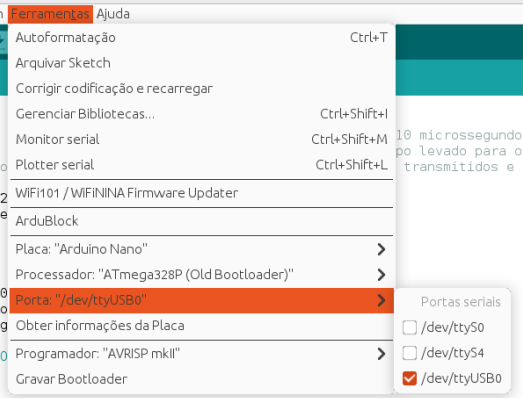
Primeiramente, explicamos ao IDE do Arduino que vamos trabalhar com o Arduino NANO:



Pode ocorrer problema de comunicação com a placa, então vamos começar selecionando a configuração para o processador mais antigo:

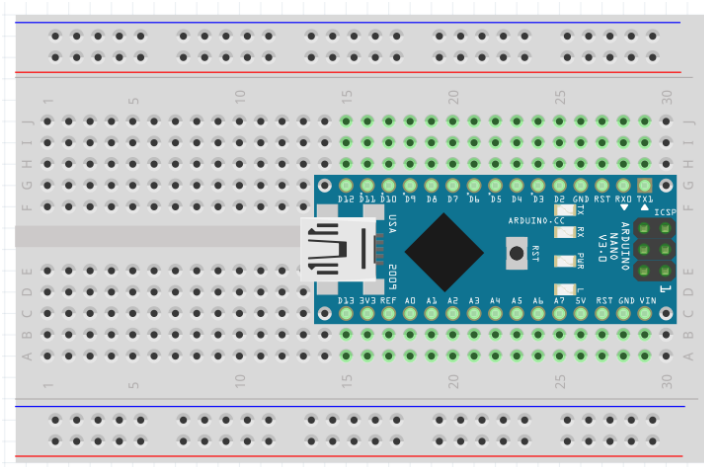


Depois, selecionamos a porta de comunicação serial que está sendo utilizada (as informações da cópia de tela que segue variam de máquina para máquina – se não souber qual a porta, no Windows dá para saber por meio do Gerenciador de Dispositivos):

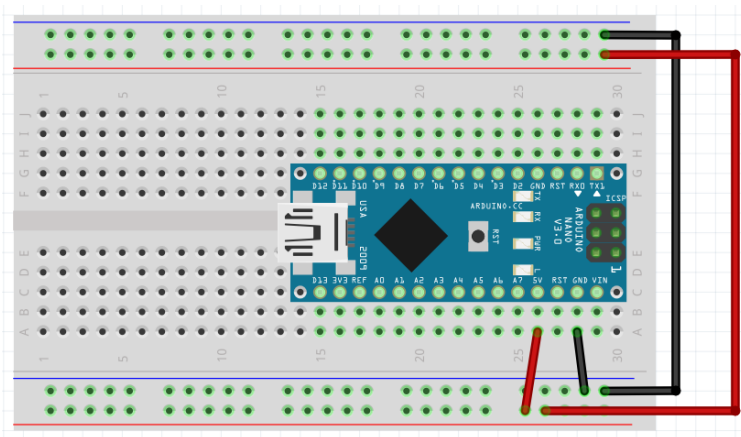


E qual código vamos desenvolver? Bem, o código depende das ligações que fizemos no hardware. Vamos fazer passo a passo.

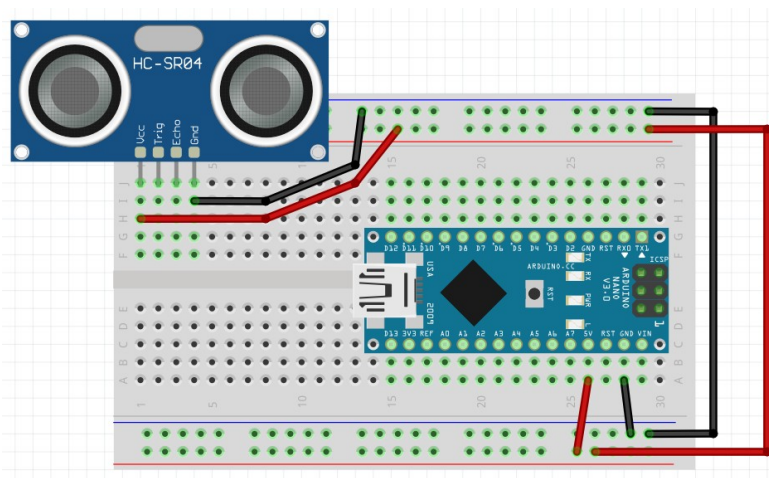
Vamos colocar a placa do Arduino NANO em uma *protoboard*:



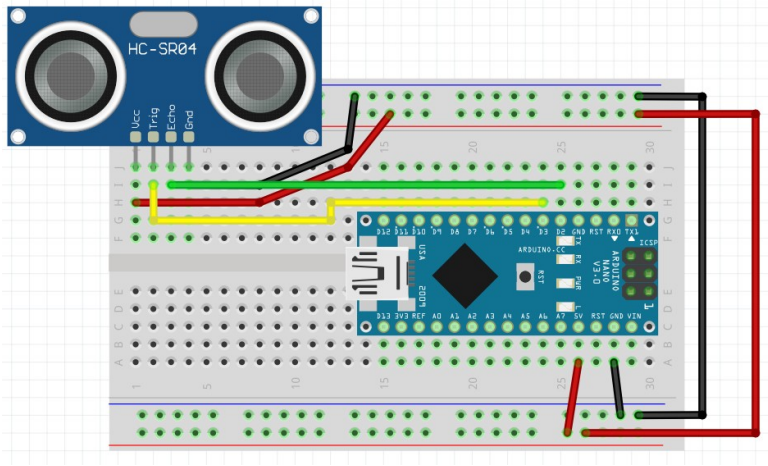
Vamos ligar as conexões de energia:



Agora, vamos colocar o sensor de ultrassom ():



Note que as conexões de energia já foram realizadas. As próximas, são as de comunicação. Podemos escolher quaisquer portas, e, depois, informaremos ao Arduino quais portas escolhemos. Por exemplo:



O ‘echo’ ficou no pino digital 2, e o ‘trigger’ ficou no pino digital 3.

Com isso, podemos testar o sensor:

```
1 /*
2 Trig é um pino de entrada do sensor, fica em nível alto (HIGH) por 10 microssegundos indicando o início da medição (após 2 us de 'reset');
3 Echo é um pino de saída do sensor, fica em nível alto durante o tempo levado para o sinal ultrassônico ir e voltar para o sensor.
4 Depois do sinal do Trig de 10 microssegundos, 8 pulsos de 40kHz são transmitidos e o receptor espera o retorno da onda.
5 */
6 #define pinoEcho 2 //Saída do sensor, entrada no Arduino
7 #define pinoTrigger 3 //entrada do sensor, saída no Arduino
8
9 void setup() {
10   Serial.begin(9600);
11   pinMode(pinoEcho, INPUT);
12   pinMode(pinoTrigger, OUTPUT);
13 }
14
15 void loop() {
16   digitalWrite(pinoTrigger, LOW);
17   delayMicroseconds(2);
18   digitalWrite(pinoTrigger, HIGH);
19   delayMicroseconds(10);
20   digitalWrite(pinoTrigger, LOW);
21   long duracao = pulseIn(pinoEcho, HIGH);
22   // velocidade do som = 331,3 + (0,606 T)
23   // 343,42 m/s a 20 graus C
24   // distancia = (343,42 * tempo) / 2 --> 343,42 m/s multiplicado pelo tempo em segundos, dividido por 2 pois o sinal tem que ir e voltar
25   // 0.034342 cm/ us
26   long medida = (0.034342 * duracao) / 2;
27   Serial.println(medida);
28   delay(1000);
29 }
```

Entre as linhas 1 e 5 temos uma pequena explicação, a qual, quando colocada entre ‘/*’ e ‘*/’, transforma-se em comentário, não interferindo na programação. O funcionamento do sensor pode ser controlado por uma ‘biblioteca’ de códigos; aqui, estamos usando programação direta: vamos enviar comandos durante um certo intervalo de tempo e, depois, teremos uma resposta.

Na linha 6 estamos explicando ao Arduino que teremos uma conexão, que será chamada de ‘pinoEcho’ e que ocorre na porta número 2; na linha 7 o mesmo ocorre para o ‘pinoTrigger’, conectado na porta 3.

A função de *setup* ocorre entre as linhas 9 e 13. Na linha 10 temos o estabelecimento do monitoramento serial, com 9600bps de velocidade (usaremos o Monitor Serial para ver o resultado de nosso código). Nas linhas 11 e 12 estamos explicando ao Arduino que uma das conexões será de entrada (INPUT) e a outra de saída (OUTPUT); tais conexões usam os ‘nomes de portas’ dados nas linhas 6 e 7.

Entre as linhas 15 e 29 temos o código que irá se repetir enquanto houver energia.

Na linha 16 colocamos o sinal de saída (para o pino de Trigger do sensor) em nível baixo (LOW) e esperamos 2 microssegundos (linha 17, `delayMicroseconds(2)`). Na sequência, na linha 18, colocamos o sinal em nível alto (HIGH), esperamos 10 microssegundos (linha 19, `delayMicroseconds(10)`) e colocamos novamente o controle em nível baixo (linha 20, `digitalWrite(pinoTrigger, LOW)`). Então, na linha 21 declaramos uma variável do tipo ‘long’, a qual receberá uma informação retornada pelo comando

'pulseIn(pinoEcho, HIGH)' – que significa que queremos saber durante quanto tempo o pino ficou em nível 'alto' (HIGH). Esta sequência é chamada de **protocolo de comunicação** (quando usamos uma biblioteca, isso fica 'escondido', não vemos como funciona).

Depois, entre as linhas 22 e 25 temos conceitos de física: a velocidade do som no ar segue a função $velocidade = 331,3 + (0,606 T)$, na qual a variável T representa a temperatura. É uma função dita de primeiro grau. Podemos calcular a velocidade, por exemplo, a uma temperatura de 20°C. Ela será de 343,42 m/s (m/s significa 'metros por segundo'). Ou seja, em um segundo, a 20°C, o som se desloca 343,42 metros.

Como estamos trabalhando com um sensor que deve medir a distância, ele deverá medir o tempo que o som, na velocidade calculada para uma certa temperatura, levou para **ir e voltar**. Ou seja, metade do tempo foi gasto com a emissão de um ultrassom, que levou certo tempo para chegar a um obstáculo, e, a outra metade foi o tempo que o som levou para voltar (o 'eco', ou reflexão). Por isso iremos dividir o tempo por 2. E, como estamos trabalhando com microssegundos, e distâncias em centímetros, vamos ajustar as unidades e usar a velocidade (20°C) como sendo de 0.034342 cm/ us (a vantagem de usar a temperatura em uma fórmula é a de que, se tivermos um sensor de temperatura, podemos ter uma medição mais precisa).

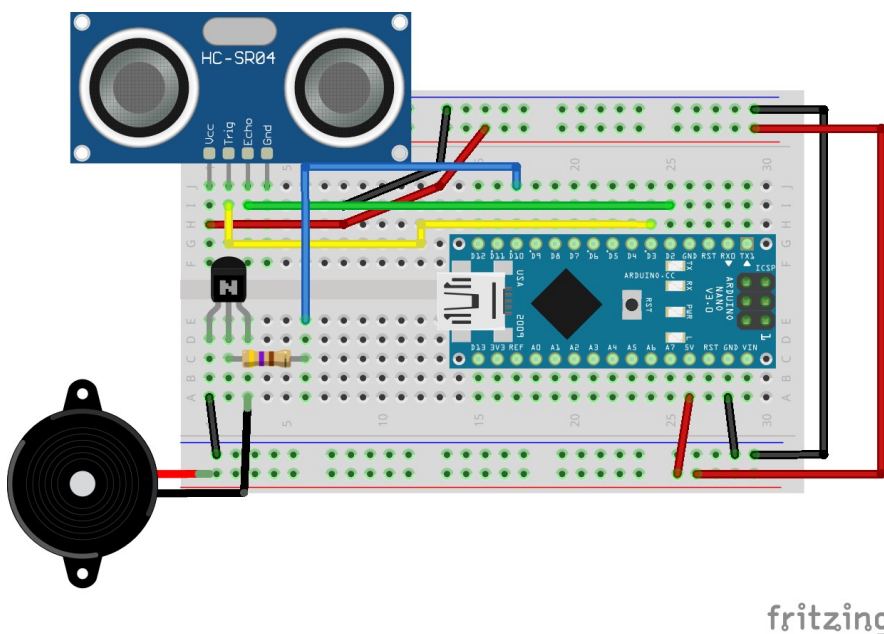
A obtenção da média é realizada na linha 26: `long medida = (0.034342 * duracao) / 2`, na qual a variável medida foi declarada como sendo do tipo de dados long, e o cálculo foi realizado multiplicando 0.034342 cm/ us pelo tempo e dividindo por 2.

Na linha 27 temos a apresentação da medida de distância realizada (em centímetros), e, na linha 28 temos uma pequena espera (de 1000 milissegundos, ou 1 segundo) antes de reiniciar a medição.

Para ver funcionando, digite os comandos (cuidado com maiúsculas e minúsculas), monte o circuito (ajustando os números de portas para sua montagem) e teste, usando Ferramentas/ Monitor Serial.

Agora que já temos o sensor de ultrassom funcionando, vamos incrementar o código. Vamos fazer com que um *buzzer* responda à distância medida: mais de 20cm fica sem som, entre 20cm e 10cm fica com um som 'baixo' e menos de 10cm fica com um som 'alto' (os valores, claro, podem ser ajustados).

Primeiro a ligação física, do *hardware*:



Ligamos um transistor NPN para ampliar o sinal e isolar o Arduino do *buzzer*. O pino 10 do Arduino foi ligado a um resistor de 470 ohms (amarelo, violeta, marrom), o qual está conectado na base do transistor. O

emissor do transistor está conectado no GND. O coletor do transistor está conectado no *buzzer*, e a outra conexão do *buzzer* é no +5V.

Será necessário adicionar código. Teremos que incluir a definição do pino do *buzzer* e os testes. Ficará assim:

```
1 /*
2 Trig é um pino de entrada do sensor, fica em nível alto (HIGH) por 10 microssegundos indicando o início da medição (após 2 us de 'reset');
3 Echo é um pino de saída do sensor, fica em nível alto durante o tempo levado para o sinal ultrassônico ir e voltar para o sensor.
4 Depois do sinal do Trig de 10 microssegundos, 8 pulsos de 40kHz são transmitidos e o receptor espera o retorno da onda.
5 */
6 #define pinoEcho 2 //Saída do sensor, entrada no Arduino
7 #define pinoTrigger 3 //entrada do sensor, saída no Arduino
8 #define buzzer 10
9
10 void setup() {
11   pinMode(pinoEcho, INPUT);
12   pinMode(pinoTrigger, OUTPUT);
13   pinMode(buzzer, OUTPUT);
14 }
15
16 void loop() {
17   digitalWrite(pinoTrigger, LOW);
18   delayMicroseconds(2);
19   digitalWrite(pinoTrigger, HIGH);
20   delayMicroseconds(10);
21   digitalWrite(pinoTrigger, LOW);
22   long duracao = pulseIn(pinoEcho, HIGH);
23   // velocidade do som = 331,3 + (0,606 T)
24   // 343,42 m/s a 20 graus C
25   // distancia = (343,42 * tempo) / 2 --> 343,42 m/s multiplicado pelo tempo em segundos, dividido por 2 pois o sinal tem que ir e voltar
26   // 0.034342 cm/ us
27   long medida = (0.034342 * duracao) / 2;
28   if (medida > 20){
29     noTone(buzzer);
30   }
31   if (medida < 20 && medida > 10){
32     tone(buzzer, 2000, 100);
33     delay(300);
34   }
35   if (medida < 10){
36     tone(buzzer, 2000, 100);
37     delay(100);
38   }
39   delay(100);
40 }
```

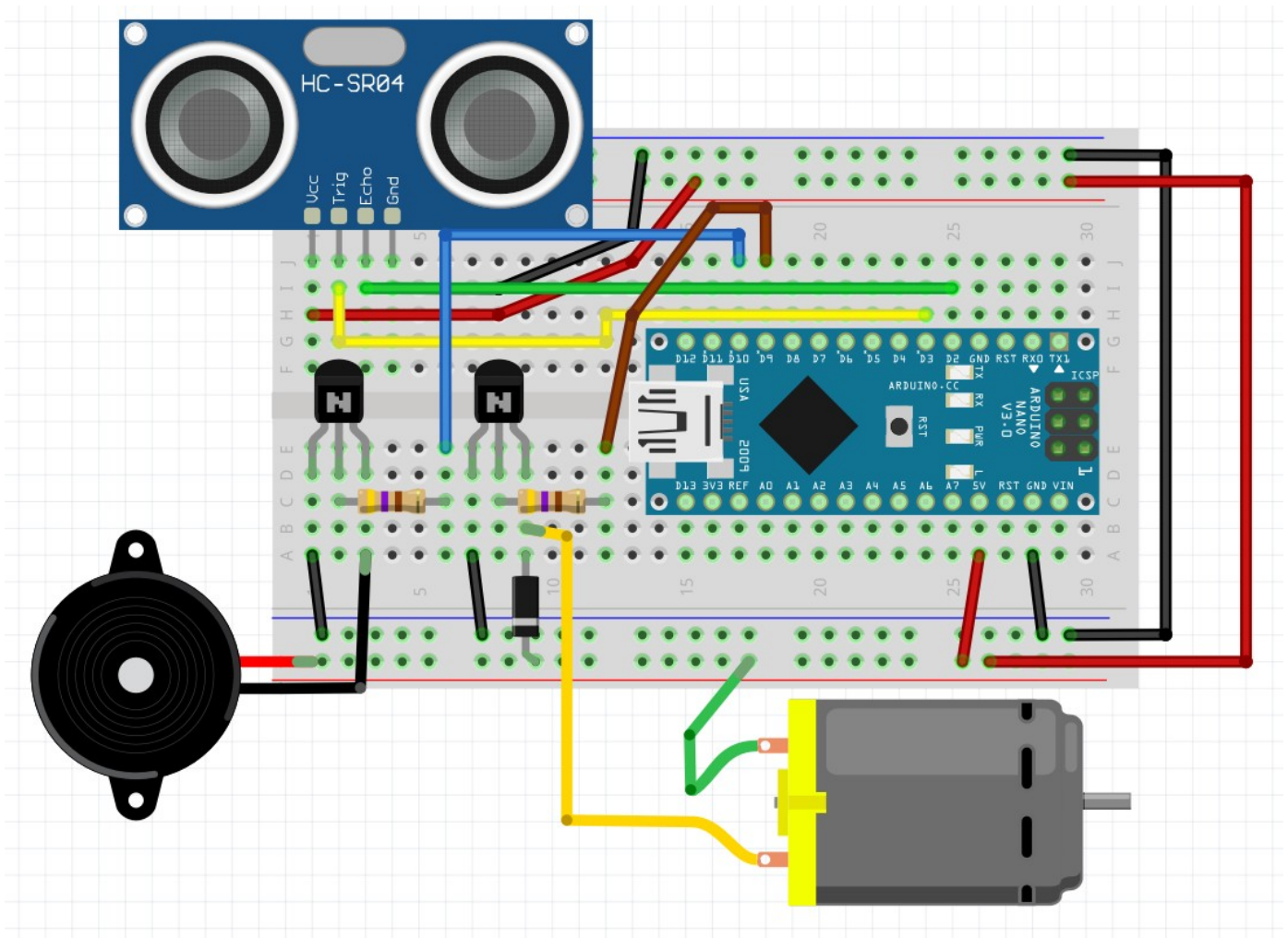
Foram inseridos novos comandos de programação: na linha 8 (definindo a ligação do *buzzer* na porta 10), na linha 13 (colocando a porta do *buzzer* no modo de saída – *OUTPUT*), e entre as linhas 28 e 38 para teste da distância. Veremos na sequência o funcionamento, mas, antes de prosseguir, note que o comando de inicialização da interface serial e os comandos de impressão da distância foram retirados, pois já temos o funcionamento garantido e não precisamos mais deles.

Vamos ver os testes. Na linha 28 temos um teste da variável ‘medida’; caso ela seja maior do que (‘>’) 20cm, não haverá nenhum som no buzzer (‘noTone’). Se o valor da variável ‘medida’ for menor do que (‘<’) 20cm e (‘&&’) for maior do que (‘>’) 10cm, serão executados os comandos dentro do bloco, as linhas 32 e 33. Na linha 32 será emitido um som com frequência 2000 e duração 100 [‘tone(buzzer, 2000, 100)’] e haverá uma pequena espera de 300 milissegundos [‘delay(300)’]. Podemos variar a frequência, a duração, e o tempo de espera para ter efeitos sonoros diferentes. A documentação do Arduino informa que ‘a função tone() pode interferir com as saídas PWM dos pinos 3 e 11, em placas que não sejam o Arduino Mega’¹.

Agora, vamos colocar um motor. Vamos usar novamente um transistor (NPN) para controlar o motor e isolar a porta do Arduino, um resistor de polarização e, para proteção do transistor, um diodo.

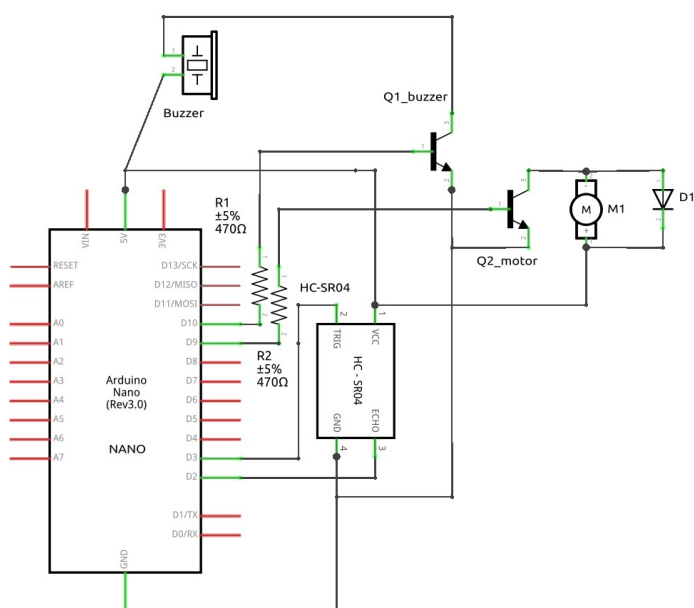
Ficará assim (a ligação do transistor de controle do motor foi realizada na porta de número 9, mas você pode escolher outra; no entanto, usaremos um controle de potência baseado em ‘PWM’, que já testamos com LEDs; desta forma, escolha uma porta PWM, que são as que começam com ‘~’ na numeração – as portas PWM são: ~3, ~5, ~6, ~9, ~10 e ~11):

1 <https://reference.arduino.cc/reference/pt/language/functions/advanced-io/tone/>



O novo transistor é conectado por meio de um resistor em sua base ao pino 9 do Arduino, o emissor é ligado ao GND. O motor é conectado entre o coletor do transistor e o +5V, com um diodo em paralelo (note que o diodo é polarizado, seu cátodo _marcado com um sinal ' | ' no invólucro_ é ligado ao +5V).

Eletricamente, temos o seguinte:



fritzing

Agora, vamos ao novo código:

```

1 /*
2 Trig é um pino de entrada do sensor, fica em nível alto (HIGH) por 10 microssegundos indicando o início da medição (após 2 us de 'reset');
3 Echo é um pino de saída do sensor, fica em nível alto durante o tempo levado para o sinal ultrassônico ir e voltar para o sensor.
4 Depois do sinal do Trig de 10 microssegundos, 8 pulsos de 40kHz são transmitidos e o receptor espera o retorno da onda.
5 */
6 #define pinoEcho 2 //Saída do sensor, entrada no Arduino
7 #define pinoTrigger 3 //entrada do sensor, saída no Arduino
8 #define buzzer 10
9 #define motor 9
10
11 void setup() {
12   pinMode(pinoEcho, INPUT);
13   pinMode(pinoTrigger, OUTPUT);
14   pinMode(buzzer, OUTPUT);
15   pinMode(motor, OUTPUT);
16 }
17
18 void loop() {
19   digitalWrite(pinoTrigger, LOW);
20   delayMicroseconds(2);
21   digitalWrite(pinoTrigger, HIGH);
22   delayMicroseconds(10);
23   digitalWrite(pinoTrigger, LOW);
24   long duracao = pulseIn(pinoEcho, HIGH);
25   // velocidade do som = 331,3 + (0,606 T)
26   // 343,42 m/s a 20 graus C
27   // distancia = (343,42 * tempo) / 2 --> 343,42 m/s multiplicado pelo tempo em segundos, dividido por 2 pois o sinal tem que ir e voltar
28   // 0.034342 cm/ us
29   long medida = (0.034342 * duracao) / 2;
30   if (medida > 20){
31     analogWrite(motor, 0);
32     noTone(buzzer);
33   }
34   if (medida < 20 && medida > 10){
35     tone(buzzer, 2000, 100);
36     mexer(100);
37     delay(300);
38   }
39   if (medida < 10){
40     tone(buzzer, 2000, 100);
41     mexer(200);
42     delay(100);
43   }
44   delay(100);
45 }
46
47 void mexer (int quanto) {
48   long horario = millis();
49   while (millis() < (horario + 300)){
50     analogWrite(motor, quanto);
51   }
52 }

```

O que mudou?

- Na linha 9, temos a declaração da porta em que o motor foi conectado: `#define motor 9`
- Na linha 15, a definição de a ligação do motor é uma saída: `pinMode(motor, OUTPUT);`
- Na linha 31, no teste para distância maior do que 20cm, foi incluída uma instrução para que o motor fique parado: `analogWrite(motor, 0);`
- Na linha 36 foi chamada a função ‘mexer’, passando como parâmetro o valor ‘100’: `mexer(100);`
- Na linha 41 foi chamada a função ‘mexer’, passando como parâmetro o valor ‘200’: `mexer(200);`
- A função ‘mexer’ foi definida entre as linhas 47 e 52 e ficou assim:

```

void mexer (int quanto) {
  long horario = millis();
  while (millis() < (horario + 300)){
    analogWrite(motor, quanto);
  }
}

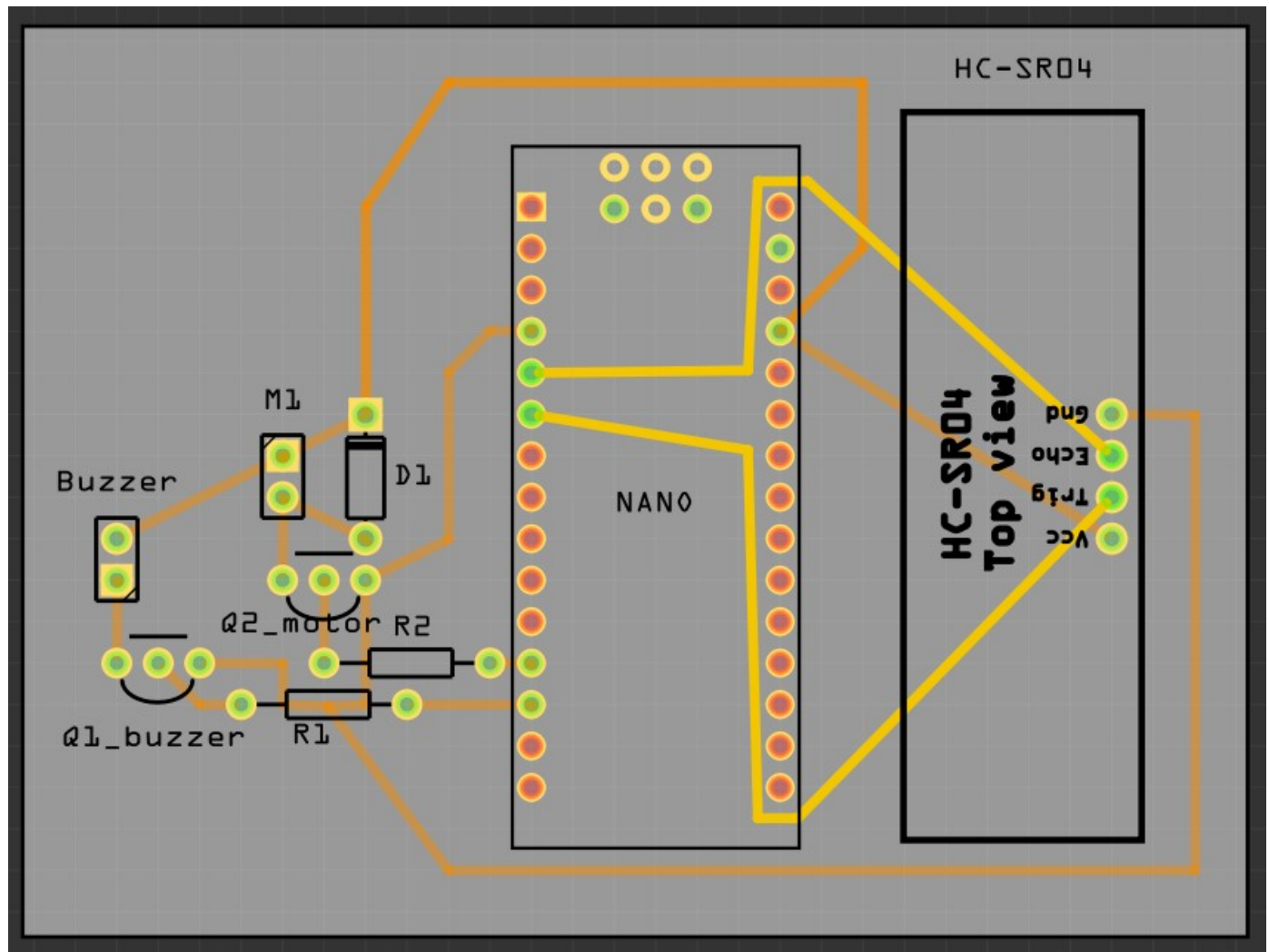
```

Esta função recebe um valor (identificado como ‘quanto’). Depois, armazena o horário por meio da função `millis()` e, em seguida, enquanto (‘while’) o horário for menor do que o horário atual mais 300 milissegundos, executa o comando de escrita analógica (‘`analogWrite`’) na porta em que está ligado o motor.

Confira as ligações e o código e teste. Pode variar as frequências, duração e tempos de espera escolhidos anotando os resultados e selecionando o mais adequado.

E, se quisermos fazer uma ‘plaquinha de circuito impresso’, assim como fizemos com o projeto do CyberPower, como fazemos o desenho dela?

Para desenhar uma PCB (*printed circuit board*, placa de circuito impresso), usamos a ‘aba’ PCB do Fritzing:



O desenho da placa pode ser ajustado (re)posicionando-se os componentes. Na figura acima, as cores de trilhas ‘laranja’ são da face inferior e as trilhas em ‘amarelo’ são da face superior. Isto supõe uma placa de duas faces. Pode não ser muito econômico usar placas de duas camadas para apenas duas trilhas superiores, então poderíamos mudar o desenho (mudando posicionamento dos componentes) ou utilizar ligações do tipo ‘*jumper*’ (pedaços de fios que fazem as conexões).

<https://tiaplicada.ufpr.br/wp-content/uploads/2023/04/nanodog.zip>