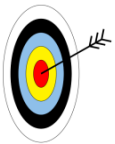


## Station 2 – Park Distance Control System – Bonus

### The Distance Creates the Sound

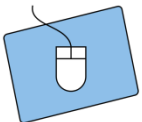
Your parking aid now measures distance values using the IR sensor, and even beeps in different speeds. However, a big disadvantage is, that you first measured the values, then manually divided the distance ranges, and also manually specified the beep speeds for these ranges. This is impractical for everyday use, because you cannot always carefully measure the distance with a vehicle in a first step, and then program the parking aid. The solution: The sensor constantly measures the change in distance, and immediately calculates the warning sound automatically.

In short: You want your park distance control system to calculate the speed of the sound directly from the measured sensor values. Unfortunately, the sensor does not directly provide centimeter readings, and these cannot be easily calculated from the values.



In the improvement, you will learn,...

- ✖ how to include, and use given functions in your sketch.
- ✖ to convert the calculated centimeter value directly into a beep speed.



1. Save your sketch, for the last time, under a meaningful name.
2. In order to be able to use the function for converting the sensor values, you have to insert a so-called library at the very beginning of the sketch. Start your program by using `#include <Abstand.h>`

**Hint:** Write this command with angle brackets but without semicolon!

**Abstand** is the German expression for **distance**. In this case, you use the German expression, because it is the name of the library, and cannot be changed.

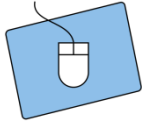


In the following, you will use a library, which contains commands with German terms. Since these commands cannot be changed easily, you will have to work with the German commands. To be sure, that you understand the signification, you will always find a translation.

### Libraries

The `include` command tells the sketch to include the library named “Abstand”, which contains the functions to convert distance values of the sensor into centimeter values. An example is the function `berechneZentimeter()` (Eng.: calculate Centimeter), which you will use later. The hash (#) in front of the command indicates, that it is a special command, which has to be executed first, and before anything else. The ending `.h` comes from the C programming language, which is used in Arduino to create such libraries.

## Station 2 – Park Distance Control System – Bonus



3. Now, you have included everything, which is necessary for the calculation. But in order to work with this library, you have to create a copy of this library. This is very close to declaring a variable. The type (e.g. `int` for numerical values) is the name of the library (i.e. `Abstand`), followed by any name.
4. Now, add an `int` variable to the settings. Here, you can later save the calculated centimeter values.
5. The real work happens in the `loop()`:

- a. The line for reading the sensor values is already in your sketch. You will now convert these large values into values between 10 and 80 cm. Therefore, you have to use the following **function**:

`berechneZentimeter(sensorValue)`

Now, you also need the **instance you created of the library**. As your sketch needs to know where the method comes from, you have to tell it through the variable.

- b. The sensor value is already stored in a variable. Here, it is sufficient to simply write the name of the variable in the **brackets**.
- c. To store the calculated centimeter values now also in a variable, you only have to connect the variable name from step 4 to the call of the method from 5a – as if you assign a simple numerical value to an `int` variable. So, you have to use an equal sign.

All in all, it looks like this:

```
sensorValue = libraryInstance.berechneZentimeter(sensorValue);
```

6. As a last step, you can now display the new centimeter values on the console. Transfer your program, and place your obstacle accordingly. Compare the values to your own measured centimeter values from the table.

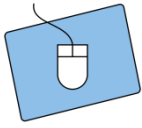
“berechneZentimeter”  
means  
“calculateCentimeter”

Distance	10 cm	20 cm	30 cm	40 cm	50 cm	60 cm	70 cm	80 cm
Sensor Value								

Now, you can finally use the centimeters, and directly determine the `delay` between the beeps! You do not need an `if` statement anymore, because you do not have to divide and query ranges by hand.

*On the next page, you go to the final spurt...*

## Station 2 – Park Distance Control System – Bonus



7. Set the **piezo** once to **high** (followed by a `delay`), and once to **low** (also followed by a `delay`). Think about, where you get the values for the delay commands.

**Hint:** Exactly this `delay` has to be determined via the calculated centimeter values. What does this mean, if you now get centimeter values between 10 and 80 cm? Adjust the values, if necessary, and test your program.

*Congratulations! You have even managed to optimize your parking aid! Have a good trip! 😊*



List of references:

👁️, ⚠️, 🖱️, 🚩 – Source: InfoSphere